MERANT™

Edition 5.1

PVCS®
Dimensions™

Developer's Toolkit Reference Guide

MERANT

3445 NW 211th Terrace

Hillsboro, Oregon 97124

Table of Contents

Welcome to Dimensions

Thank you for choosing MERANT™ PVCS® Dimensions™, a powerful process management and change control system that will revolutionize the way you develop software. Dimensions helps you organize, manage, and protect your software development projects on every level—from storing and tracking changes to individual files, to managing and monitoring an entire development cycle.

Purpose of this manual

The purpose of this manual is to detail how you use the Dimensions Developer's Toolkit (DTK) to access and manipulate objects that are held within a PVCS Dimensions repository. This document covers descriptions of the interface functions that the DTK provides, and details on the Event Callout Interface that enables you to perform customizations and integrations around Dimensions commands.

The intended audience is users who are well versed in both Dimensions concepts and the C programming language.

For more information

Refer to the *PVCS Dimensions Getting Started Guide* for a description of the Dimensions documentation set, a summary of the ways to work with Dimensions, and instructions for accessing the Online Help.

Edition status

This is Edition *5.1* of the *PVCS Dimensions Developer's Toolkit Reference Guide.* The information in this edition applies to *Release 7.1 of PVCS Dimensions* or later. This edition supersedes earlier editions of this manual.

Typographical Conventions

The following typographical conventions are used in the online manuals and online help. These typographical conventions are used to assist you when using the documentation; they are not meant to contradict or change any standard use of typographical conventions in the various product components or the host operating system.

Convention	Explanation
italics	Introduces new terms that you may not be familiar with and occasionally indicates emphasis.
bold	Emphasizes important information and field names.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
`monospace`	Indicates syntax examples, values that you specify, or results that you receive.
`monospaced italics`	Indicates names that are placeholders for values you specify; for example, *`filename`*.
`monospace bold`	Indicates the results of an executed command.
vertical rule |	Separates menus and their associated commands. For example, select File | Copy means to select Copy from the File menu. Also, indicates mutually exclusive choices in a command syntax line.
brackets []	Indicates optional items. For example, in the following statement: `SELECT [DISTINCT]`, DISTINCT is an optional keyword.
. . .	Indicates command arguments that can have more than one value.

Ordering Hard-Copy Manuals

As part of your Dimensions license agreement, you may print and distribute as many copies of the PVCS Dimensions manuals as needed.

If you do not want to print each of these online manuals, you can order hard-copy versions from MERANT. To order, please contact your sales representative for assistance.

Contacting Technical Support

MERANT provides technical support for all registered users of this product, including limited installation support for the first 30 days. If you need support after that time, contact MERANT using one of the methods listed in any of the *Installation Guides*, the *Getting Started Guide,* or the Online Help.

Technical support is available 24 hours a day, 7 days a week, with language-specific support available during local business hours. For all other hours, technical support is provided in English.

Support via the web, E-mail, and telephone

SupportNet Customers can report problems and ask questions on the SupportNet web page: `http://support.merant.com/`

To submit an issue, click on the **Report a Problem** link and follow the instructions. You can also submit issues via E-mail or phone. Refer to the *Installation Guides*, *Getting Started Guide*, or Online help for a list of contact numbers, including numbers to call for local language support.

The SupportNet Web site contains up-to-date technical support information. Our SupportNet Community shares information via the Web, automatic E-mail notification, newsgroups, and regional user groups.

SupportNet Online is our global service network that provides access to valuable tools and information for an online community for users. SupportNet Online also includes a KnowledgeBase, which contains how-to information and allows you to search on keywords for technical bulletins. You can also download fix releases for your PVCS products.

1 What is the Dimensions Toolkit Interface?

In this Chapter

Overview

The Dimensions Developer's Toolkit Interface (DTK) is a powerful C and C++ Applications Programming Interface (API) that allows you to access data held within a PVCS Dimensions repository.

The DTK provides a way in which you can:

- design and implement your own applications that can interact with Dimensions
- implement your own specific customizations using a rich event callout interface.

This chapter takes you through the architecture of the DTK and how you can use it to expand on the functionality offered by Dimensions.

Positioning Your Solutions within the DTK

The DTK provides two comprehensive architectures that allows you to integrate your solutions with Dimensions in the following ways:

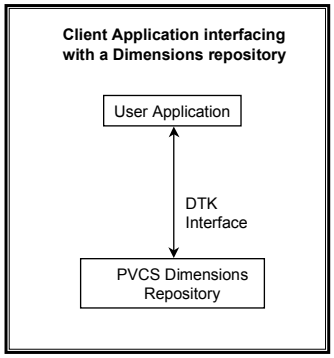
1. As a separate client application that uses the DTK to access and manipulate objects held within a Dimensions repository
2. As a rich event callout interface that allows you to perform your own customized operations when certain Dimensions commands are run.

When you are looking at your requirements keep in mind where in the DTK architecture you wish to position your solution. If, for example, you have a requirement where you need to assess the impact to one of your projects of implementing a number of application changes (as cited in change documents), then you would use the format for Client Architecture as stated in the following subsection.

Client Architecture

Using the *Client Architecture* you would design an application that would:

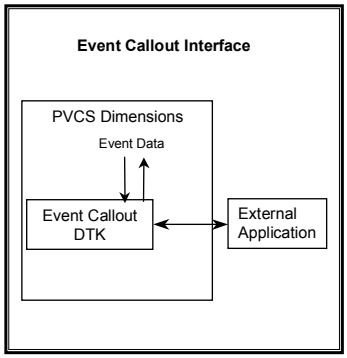
- connect to your Dimensions repository
- query or manipulate the data from that repository
- disconnect from the repository and take some action based on that data.



This scheme enables you to use the DTK as an I/O interface into the Dimensions repository. If, however, you wanted a specific operation or action to be performed when certain Dimensions commands are run, then you would use the *Event Architecture* described below.

Event Architecture

Using the *Event Architecture* you can design a set of customizations that are applied explicitly when a user issues a certain Dimensions command. A public interface is provided that allows you to pass information back and forth between the event and the Dimensions server. The result of this is that you can manipulate the outcome of the command in certain ways.



For more details on how events operate, please refer to Chapter 2, “Writing Dimensions DTK Applications,” on page 19.

Interactions between the Two Architectures

When you design and implement a customization, using the *Event Architecture*, this customization is applied to all the standard Dimensions interfaces. This customization is also applied to all those applications that you have developed using the *Client Architecture.* Events, when they are deployed, literally become part of the Dimensions product suite, and as a result are used by all Dimensions components.

Scope of the DTK Architecture

When you look at designing your applications for either of the specific architectures described above, there are a number of points that you must keep in mind:

- **DTK Applications using the Client Architecture**

  These applications are stand-alone utilities that interface with the Dimensions repository via the use of *PcmsConnect()* and other DTK functions.

- **DTK Applications using the Event Architecture**

  These applications interface with Dimensions via a public C function call named *userSuppliedFunction().* These applications are built as shared libraries that are dynamically loaded by Dimensions. These applications are able to share information with the Dimensions server and so do not need to call *PcmsConnect()* or *PcmsDisconnect()*.

  For events to become active they only need to be deployed on the Dimensions server. As a result of this, each client which accesses that Dimensions server will use these events. You do not need to be concerned with deploying and controlling events on multiple client installations.

2 Writing Dimensions DTK Applications

In this Chapter

Introduction

This chapter outlines the data structures, manifest constants and return codes that are used by the DTK. These structures and constants are defined in the provided include file *pcms_api.h* in the directory:

- "*<Dimensions_ROOT>/pcms_api/*" for UNIX
- "*<Dimensions_ROOT>**pcms_api*\" for Windows.

Any source file which references DTK functions or constants must include this file.

NOTE Starting with Dimensions 7.1, the pcms_api.lib and pcms_api.so library files have been renamed to pcms_api*XX*.lib and pcms_api*XX*.so, where *XX* is the version number of the Dimensions release. For example, for Dimensions 7.1, the files are named pcms_api71.lib and pcms_api71.so.

DTK Return Codes

In general when you call a DTK function the results of that function call can be determined by the return code given. There are three codes that a DTK function can return:

PCMS_OK	which indicates that the function call succeeded, and objects were processed (e.g. a query returned some information).

PCMS_FAIL which indicates that while the function call succeeded, no objects were actually processed (e.g. a query returned no objects).

PCMS_ERROR which indicates that an error occurred trying to process the function call. If an error is encountered, the reason for the error can be assessed by examining the following variables.

int char	*PcmsErrorNo* **PcmsErrorStr*	If the error occurred due to a programming error e.g. invalid parameters were specified to a DTK function, these variables will be set detailing the reason for the error.
int char	*PcmsDbErrorNo* **PcmsDbErrorStr*	If the error occurred due to a database error e.g. the database is full, these variables will be set detailing the reason for the error. *PcmsDbErrorNo* will indicate the database error number (if any). *PcmsDbErrorStr* will indicate the SQL error string.

NOTE *ErrorNo* variables and *ErrorStr* variables will be set only when an error occurs.

DTK Data Structures

The results of function calls are generally stored in the data structures that are defined in the following sub-sections. These structures represent the abstraction of Dimensions objects and other information, and can be accessed via standard C constructions.

PcmsObjStruct

Definition

```
typedef struct
{
        int     uid;
        int     objType;        /* PCMS_ITEM or PCMS_PART or */
                                /* PCMS_CHDOC etc */
        int     typeUid;
        char    typeName[PCMS_L_TYPE_NAME + 1];
        char    productId[PCMS_L_PRODUCT_ID + 1];
        char    objId[PCMS_L_CD_ID + 1];
        char    variant[PCMS_L_VARIANT + 1];
        char    revision[PCMS_L_REVISION + 1];
        char    description[PCMS_L_DESCRIPTION + 1];
        char    userName[PCMS_L_USER + 1];
        char    status[PCMS_L_STATUS + 1];
        char    dateTime[PCMS_L_DATE_TIME + 1];
        char    isExtracted;    /* 'Y' = Yes, 'N' = No */
        int     noAttrs;        /* The number of pcms_defined*/
                                /* attributes for this object */
        PcmsObjAttrStruct  *attrs;
                        /* Pointer to the array of*/
                        /* pcms_defined attributes*/
} PcmsObjStruct;
```

Description

This is the generic structure used for Dimensions objects such as items, parts, change documents and baselines. The type of object is defined by the member field *objType* being set to a specific constant (e.g. PCMS_ITEM). The **attrs* pointer can be used to access attribute information if it is defined.

PcmsCallbackStruct

Definition

```
typedef struct PcmsCallbackStruct
{
    PcmsCallbackProc callback;
    void *clientData;
}   PcmsCallbackStruct;
```

Description

This structure is used to hold information regarding registered callbacks. For more information please refer to *PcmsSetCallback().*

PcmsObjAttrStruct

Definition

```
typedef struct
{
        int     attr;           /* Attribute number */
        void    *value;         /* The value of the attribute. */
                                /* See section on Attribute */
                                /* Macros */
        PcmsObjAttrDefStruct  *attrDef;
                                /* Pointer to the definition */
                                /* of the attribute */
} PcmsObjAttrStruct;
```

Description

This structure is used to hold information regarding the attributes that an object has. The *attr* member details the attribute number, while the **attrDef* pointer contains details on the attribute definition. The value of the attribute is accessed via the *PcmsMvaGetVal()* and *PcmsSvaGetVal()* attribute macros. For more information about these macros please refer to the "Attribute Macros" on page 139.

PcmsObjAttrDefStruct

Definition

```
typedef struct
{
        int     attr;           /* Attribute number */
        int     valueMaxLen;    /* The maximum length of the */
                                /* attribute */
        char    variable [PCMS_L_ATTR_VARIABLE + 1];
                                /*The attribute name */
        char    prompt [PCMS_L_ATTR_PROMPT + 1];
                                /* The screen prompt*/
        char    attrType;       /* PCMS_ATTR_DATE = 'D' */
                                /* PCMS_ATTR_UNDEFINED = 'U' */
                                /* PCMS_ATTR_INTEGER = 'I' */
                                /* PCMS_ATTR_NUMBER = 'N' */
                                /* PCMS_ATTR_CHAR = 'C' */
        char    scope;          /* PCMS_ATTR_ITEM='I' */
                                /* PCMS_ATTR_PART='P'*/
        char    display;        /* Y or N */
        char    allRevisions;   /* Y or N */
        char    manOpt;         /* MANDATORY = 'Y' */
                                /* OPTIONAL = 'N' */
        char    fldUpd;         /* 'Y' or 'N' */
        char    roleCheck [PCMS_L_ROLE + 1];
        char    uniqueVal;      /* 'Y' *or 'N' /
        char    defaultVal[PCMS_L_ATTR_DEFAULT_VAL + 1];
        char    helpMess [PCMS_L_ATTR_HELP_MESS + 1];
        char    validationOn;   /* Is validation enabled, */
                                /* Y or N */
        int     definedBy;      /* PCMS_ATTR_PCMS/PROG/USER */
        char    hasLov;         /* 'Y' or 'N'. List of Values */
                                /* must be used to set */
        void    **pp;           /* Reserved */
        char    mva;            /* 'Y' or 'N'.Multi-Valued */
                                /* use PcmsMva... macros to */
                                /* interpret. * /
        char    mvaType;        /* not used currently */
        char    blockName [PCMS_L_ID + 1];
                                /* attr may belong to a */
                                /* display Block */
        int     blockColNo;     /* Column number in the */
                                /* display block */
        int     displayWidth;   /* Recommended display width */
        int     displayHeight; /* Recommended display height */
        char    multiLine;
                        /* 'Y' - use displayHeight ie. */
                        /* displayHeight > 0 */
                        /* 'N' - displayHeight not used */
```

continued

```
        char    valueCase;      /* 'L' - Lower 'U' - Upper, or */
                                /* 'M' - Mixed */
        char    catalogDisplay;         /* 'Y' or 'N' */
} PcmsObjAttrDefStruct;
```

Description

This structure is used to hold information relating to the attribute definition. Some of the fields for change documents are currently hard-wired to the following values.

allRevisions	'N'
manOpt'	'N'
fldUpd	'Y'
roleCheck	'\0'
uniqueVal	'N'

PcmsRelTypeStruct

Definition

```
typedef struct
{
        int     uid;    /* of this relationship subType */
        char    name [ PCMS_L_ID + 1];
                        /* Name for this rel subtype*/
        int     relType;
                        /* The parent relationship type */
                        /* eg.PCMS_REL_INFO */
        char    productId [ PCMS_L_PRODUCT_ID  + 1];
                        /* Product name */
} PcmsRelTypeStruct;
```

Description

This structure is used to hold information relating to the user-defined types used in Dimensions. For example, ’change document to change document’ relationships.

PcmsRelStruct

Definition

```
typedef struct
{
        int     uid;
        int     objType;
        int     relType;
        int     relSubTypeUid;
                        /* Refers to the uid field of a */
                        /* PcmsRelTypeStruct.  For chdocs, */
                        /* users may setup up specialization's */
                        /* of the basic relTypes to add */
                        /* attributes,etc. See section on */
                        /* PcmsGetUserRels */
} PcmsRelStruct;
```

Description

This structure is used in conjunction with *PcmsRelTypeStruct* and stores the relationships that an object has.

PcmsUserRoleStruct

Definition

```
typedef struct
{
        char    user [PCMS_L_USER + 1];       /* User */
        char    role [PCMS_L_ROLE + 1];       /* Role */
        char    capability;     /* The users capability in */
                                /* this role, either */
                                /* 'P' - Primary or */
                                /* 'S' - Secondary or */
                                /* 'L' - Leader */
        char    applyDeny;      /* applyDeny flag not */
                                /* currently used */
        char    treeWalk;       /* treeWalk flag not */
                                /* currently used */
        char    actionable;
                                /* PCMS_ACT_NOT_LEADER = '1' */
                                /* (Can't action not leader) */
                                /* PCMS_ACT_OK = '2' */
                                /* (Can action no leaders) */
                                /* PCMS_ACT_LEADER = '3' */
                                /* (Can action I am a leader) */
        char    fromTree;       /* 'Y' This role was found */
                                /* from the Part Structure */
                                /* 'N' This role was delegated */
                                /* using the DLGC command */
} PcmsUserRoleStruct;
```

Description

This structure is used to hold information relating to role assignments.

PcmsPendingUserStruct

Definition

```
typedef struct
{
        char    user [PCMS_L_USER+ 1];        /* User */
        char    role [PCMS_L_ROLE + 1];       /* Role */
        char    capability;     /* The user's capability in */
                                /* role, either */
                                /* 'P' - Primary or */
                                /* 'S' - Secondary or */
                                /* 'L' – Leader */
        char    nextStatus [PCMS_L_STATUS + 1];
                                /* next possible status */
        int     nextPhase;      /* next phase */
        char    actionable;
                                /* PCMS_ACT_NOT_LEADER = '1' */
                                /* (Can't action not leader) */
                                /* PCMS_ACT_OK = '2' */
                                /* (Can action no leaders) */
                                /* PCMS_ACT_LEADER = '3' */
                                /* (Can action I am a leader) */
} PcmsPendingUserStruct;
```

Description

This structure holds information relating to who can action an object and to what states.

PcmsRoleStruct

Definition

```
typedef struct
{
        char    role [PCMS_L_ROLE + 1];       /* Role */
        int     uid;                    /* uid of the part */
        char    leader;         /* 'Y'  = Leader only role */
} PcmsRoleStruct;
```

Description

This structure holds information of the roles defined on a Dimensions product.

PcmsLcStruct

Definition

```
typedef struct
{
        char    normalPath;
        int     phase;
        char    status [PCMS_L_STATUS + 1];
        char    role [PCMS_L_ROLE + 1];
} PcmsLcStruct;
```

Description

This structure holds lifecycle definition information.

PcmsTypeStruct

Definition

```
typedef struct
{
        int     uid;
        char    productId [PCMS_L_PRODUCT_ID + 1];
        char    typeName [PCMS_L_TYPE_NAME + 1];
        int     objType;        /* PCMS_PART, PCMS_ITEM, */
                                /* PCMS_CHDOC etc*/
        char    lifecycle [PCMS_L_ID + 1];
        char    typeDescription [PCMS_L_DESCRIPTION + 1];
        char    superType;      /* '1', '2', '3', or '4' for */
                                /* chdocs only */
} PcmsTypeStruct;
```

Description

This structure holds object type definition information.

PcmsPendStruct

Definition

```
typedef struct
{
        int     objUid;
        int     objType;
        char    capability;     /* 'P' - Primary, */
                                /* 'S'- Secondary, */
                                /* 'L' - Leader*/
        char    actionable;     /* PCMS_ACT_NOT_LEADER = '1'*/
                                /* (can't action not leader), */
                                /* PCMS_ACT_OK = '2' */
                                /* (Can action no leaders), */
                                /* PCMS_ACT_LEADER = '3' */
                                /* (Can action I am a leader) */
} PcmsPendStruct;
```

Description

This structure holds object pending information.

PcmsEventStruct

Definition

```
typedef struct
{
        char    *database;      /* PCMS ORACLE database */
                                /* identification */
        char    *baseDB;        /* PCMS Base Database */
        int     eventId;        /* PCMS_EVENT_XXX */
                                /* See userSuppliedFunction */
        int     objType;        /* PCMS_ITEM, PCMS_PART, */
                                /* PCMS_CHDOC */
        int     noAttrsChanged;   /*The number of attributes */
                                  /* that changed */
        PcmsObjAttrStruct  *attrsChanged;
                                /* Attribute number of field */
                                /* that changed */
        int     whenCalled;     /* PCMS_EVENT_VALIDATE_OP, */
                                /* PCMS_EVENT_PRE_OP or */
                                /* PCMS_EVENT_POST_OP */
} PcmsEventStruct;
```

Description

This structure is used exclusively for events and defines which event is being fired and with what parameters.

DTK System Attribute Definitions

A Dimensions object can have two kinds of attributes:

- User-defined attributes

  These are attributes defined by you in the process model.

- System-defined attributes

  These are attributes definitions that are internal to Dimensions. These attributes are provided to allow you to access information that might be useful.

The table below details the system attributes that are available for each object type.

NOTE The number of system-defined attributes are defined by the constant `PCMS_NUM_<objtype>_ATTRS`
e.g. *`PCMS_NUM_ITEM_ATTRS`*.

Object Type PCMS_ :	Number of System Attributes PCMS_NUM_ :	Attribute Reference PCMS_ATTR_ :	Definition
BASELINE	BLN_ATTRS	TEMPLATE	Template name used to create the baseline
		BASELINE_TYPE	The type of baseline created 1 = Design 2 = Release 3 = Archive
		SENDER_ID	This attribute is only populated if this baseline was created as a result of replication. This will correspond to the database that sent the baseline.
		BASELINE_METHOD	Indicates how the baseline was created. Possible values are: ■ Created = Created via CBL ■ Revised = Created via CRB ■ Merged = Created via CMB
		CREATE_DATE	Create date of the baseline
CHDOC	CHD_ATTRS	CHSEQ	Sequence number of the change document
		CREATE_DATE	Create date of the change document

Object Type PCMS_ :	Number of System Attributes PCMS_NUM_ :	Attribute Reference PCMS_ATTR_ :	Definition
		ORIGINATOR	Originator of the change document
		NO_ACTIONS	Number of actions on the change document
		SUPER_TYPE	Super type of the change document
		PHASE	Current phase of the change document
		UPDATE_DATE	Update date of the change document
		LIFECYCLE	Lifecycle assigned to the change document
ITEM	ITEM_ATTRS	FORMAT	Item format
		FILENAME	Workset filename (current workset)
		FILE_VERSION	File version in the library
		ITEM_SPEC_UID	Item spec uid
		DIR_UID	Workset directory uid
		LIB_FILENAME	Library item filename
		LIB_CHECKSUM	Library item file checksum
		LIB_FILE_LENGTH	Length of the item file in the library
		CHECKSUM	Checksum of the workset file
		SHARED_BRANCH	This attribute is reserved for future use
		FILE_LENGTH	Length of the workset file

Object Type PCMS_ :	Number of System Attributes PCMS_NUM_ :	Attribute Reference PCMS_ATTR_ :	Definition
		EDITABLE	If the item is editable
		COMPRESSED	If the item is compressed
		DIRPATH	Workset directory path
		USER_FILENAME	The user filename resulting from check out (extract), update, check in (return), etc., operations
		REVISED_DATE	Item's last revised date (Julian date format)
		CREATE_DATE	Date of item creation
		ORIGINATOR	Who created the item
		STATUS	Status of the item
		PHASE	The item's current phase
		LIFECYCLE	Lifecycle id that is followed by the item
		SENDER_ID	This attribute is only populated if this item was created as a result of replication. If an item has been remotely replicated, then this will correspond to the database that sent the item. If an item has been replicated locally, then this will correspond to the replication configuration identifier.
PART	PART_ATTRS	PARTNO	Part Number

Object Type PCMS_ :	Number of System Attributes PCMS_NUM_ :	Attribute Reference PCMS_ATTR_ :	Definition
		LOCALNO	Local Part Number
		PHASE	Part Phase
WORKSET	WORKSET_ATTRS	WS_DIR	User's default workset directory
		TRUNK	If this is a trunk workset
		ENFORCE_REV	If revision generation is enforced
		PHASE	The workset's current phase
USER	USER_ATTRS	GROUP FULL_USERNAME PHONE DEPT SITE	Additional user properties which are defined when the user is created.

DTK Constant Definitions

The following constant definitions are used within the DTK to represent Dimensions objects, relationships and phases.

- **Object types (i.e. objType)**

  Object types are constants used in the DTK to indicate classes of objects (e.g. items).

Constant	Object Definition
PCMS_BASELINE	Dimensions Baselines
PCMS_CHDOC	Dimensions Change Document
PCMS_CUSTOMER	Dimensions Customers
PCMS_ITEM	Dimensions Items
PCMS_PART	Dimensions Design Parts
PCMS_USER	Dimensions Users
PCMS_WORKSET	Dimensions Worksets

- **Relationships (i.e. relType)**

  Relationship types are constants used in the DTK to indicate relationships between objects (e.g. usage).

Constant	Relationship Definition
PCMS_REL_AFF	Affected
PCMS_REL_BREAKDOWN	Owner
PCMS_REL_DEP	Dependent
PCMS_REL_DERIVED	Built item
PCMS_REL_INFO	Information
PCMS_REL_IRT	In Response To
PCMS_REL_OWN	Same as for PCMS_REL_BREAKDOWN

continued

Constant	Relationship Definition
PCMS_REL_PRED	Predecessor
PCMS_REL_SUCC	Successor
PCMS_REL_TOP	Top owner object
PCMS_REL_USE	Usage

- **Phases**

Phases are generic indicators that are used to show where a particular object is in its lifecycle. Please refer to the related document “PVCS® Dimensions™ - Concepts Guide” for more information.

Memory Allocation within the DTK

Some platforms, such as Windows NT, require that the shared library that allocated memory is also responsible for freeing that memory. Because of this a number of wrapper functions to the standard C memory functions have been provided.

DTK Function	Wrapper to function
PcmsEvntFree()	free()
PcmsEvntMalloc()	malloc()
PcmsEvntCalloc()	calloc()
PcmsEvntRealloc()	realloc()

Usage of the Functions

- General Usage of these Functions to Allocate Memory

  If any application, either in events or clients, requires you to allocate memory that will be used by DTK functions, then you must use the wrappers as listed above. For example,

PcmsQuery() allows you to dynamically allocate memory to the *PcmsObjStruct attrs* pointer to define user-defined filters. This memory must be allocated and re-allocated via the use of the wrappers listed above. If you are allocating memory which is not used by the DTK, then you do not have to use these wrappers. However, it is strongly recommended for consistency that you use these wrappers for any memory allocation that you make.

- General Usage of these Functions to De-allocate Memory

  If any memory has been allocated by the DTK, or by the wrapper functions described above, then this memory must be freed via *PcmsEvntFree().*

- Within DTK client applications, any memory that has been allocated by DTK functions, such as *PcmsQuery(),* must be freed by the function *PcmsEvntFree()*. For example, if you have the following call:

```
int*uids = 0;
intnoUids = 0;

if (PcmsQuery(conId, &queryObj,0,&noUIds,&uids)!=PCMS_OK)
   PcmsEvntFree(uids);
```

  then you would use *PcmsEvntFree()* function to free this memory. You do not need to use these functions if you are allocating memory which is not used by the DTK. However, to be consistent in the memory functions which you do use, it is strongly recommended that you use these functions for all memory allocation and de-allocation.

- Within DTK events these functions must always be used to allocate or de-allocate memory. This includes both memory usage with events and with reference to DTK function calls, for example:

```
char  *txt = (char *)PcmsEvntMalloc(15);

status = PcmsQuery(conId, &queryObj,0,&noUIds,&uids);
PcmsEvntFree(uids);
PcmsEvntFree(txt);
```

To minimize the impact of any code changes, it is suggested that you redefine the standard C functions to use the new functions via the use of *#define*(s), for example:

```
#define      free            PcmsEvntFree
#define      malloc          PcmsEvntMalloc
....
```

and then recompile the event code. Please note, however, that the prototype of the function *PcmsEventCalloc()* is not the same as *calloc().* Please consult Chapter 3 for more information.

NOTE These functions must always be used when freeing memory that has been allocated by DTK function calls (such as *PcmsQuery()*) both within events and within DTK client programs.

The table shown below is provided as a guideline to help you identify which functions and pointers are dynamically assigned memory by the DTK, and what functions you should use to free that memory. Ensure that you refer to the DTK function description for more information on when this memory may be assigned.

Function Name	Pointer to Free	Free via
PcmsAttrDefInit()	PcmsObjAttrDefStruct *ptrDef	PcmsEvntFree()
PcmsAttrGetLov()	char ***ptrVal char **ptrMess	PcmsLovFree() PcmsEvntFree()
PcmsCntrlPlanGet()	void **ptr	PcmsEvntFree()
PcmsExecCommand()	char **ptrResponse	PcmsEvntFree()
PcmsFullQuery()	PcmsObjStruct **ptrObjs	PcmsObjFree()
PcmsGetAttrFile()	char **ptrFile	PcmsEvntFree()
PcmsGetAttrs()	PcmsObjStruct *ptrObj	PcmsObjFree()
PcmsGetCandidates()	char ***ptrCans	PcmsEvntFree()
PcmsGetPendingUsers()	PcmsPendingUserStruct **ptrUsers	PcmsEvntFree()
PcmsGetRSAttrs()	int **ptrAttrs char **ptrDefRole	PcmsEvntFree()

Function Name	Pointer to Free	Free via

continued

PcmsGetRSNames()	char **ptrMessage char ***ptrVal	PcmsEvntFree()
PcmsGetUserRelTypes()	PcmsRelTypeStruct **ptrRels	PcmsEvntFree()
PcmsGetUserRoles()	PcmsGetUserRoles **ptrRoles	PcmsEvntFree()
PcmsObjGetBackRels()	PcmsRelStruct **ptrRels	PcmsEvntFree()
PcmsObjGetRels()	PcmsRelStruct **ptrRels	PcmsEvntFree()
PcmsPendGet()	PcmsPendStruct **ptrPend	PcmsEvntFree()
PcmsPendWhoGet()	PcmsPendStruct **ptrPend	PcmsEvntFree()
PcmsPopulate()	PcmsObjStruct **ptrObj	PcmsObjFree()
PcmsQuery()	int **uids	PcmsEvntFree()
PcmsClntApiGetLastErrorEx()	char **errorBuffer	PcmsClntApiFree()

3 DTK API Functions for C/C++

In this Chapter

Introduction

This chapter describes each of the functions that are available in the DTK for C/C++ programs. The description of each function has the following components.

Purpose	What the function does
Prototype	The function prototype
Parameters	Description of the parameters used in the function
Return Codes	Codes returned (please refer to "DTK Return Codes" on page 20 for further details)
Sample	Sample function call (if applicable)
Comments	Additional relevant information (if applicable)
Related Functions	Any related DTK function calls

Before reading this chapter ensure that you have familiarized yourself with the contents of the previous chapter because it contains important information relating to data structures, return codes and manifest constants that are referenced in this chapter.

Memory Allocation by DTK Functions

A number of the DTK functions allocate memory to pointers that then becomes the responsibility of the calling application to free. On some operating systems, such as Windows NT and Solaris, memory that has been allocated by a shared library must be freed by that same shared library. You must free this memory via the function call *PcmsEvntFree().* If you do not use this function, you may experience memory corruption. For more information please refer to page 37.

PcmsSetIdleChecker - Install Idle Checker

Purpose

This function installs an application function to be called before any Dimensions DTK functions are blocked on a read. *PcmsSetIdleChecker()* will be called before making a connection to a database, and may be useful in X applications to process events from the X event queue while the Dimensions DTK is waiting for input.

Prototype

```
int
PcmsSetIdleChecker     (
       int             (*userIdleChecker )(int fd, int flag)
);
```

Parameters

userIdleChecker	is the address of the application function which will be called.

Return Codes

PcmsSetIdleChecker() returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo, PcmsErrorStr, PcmsDbErrorNo,* and *PcmsDbErrorStr.*

Comments

The prototype of the idle checker function is:

```
int idleCheckerFunction(int fd, int flag)
```

where

fd	is the file descriptor for the data connection that is about to be read
flag	is either PCMS_MSG_WAIT, or PCMS_MSG_NOWAIT.

If the flag is set to PCMS_MSG_WAIT, the function should only return when data is available to be read from the file descriptor, in which case the return value will be PCMS_OK. If the flag is set to PCMS_MSG_NOWAIT, the function will return immediately with a return value of PCMS_OK if data is available, and a return value of PCMS_FAIL if no data is available.

For X applications, *XtAddInput()* can be used to set a callback procedure when input is pending on *fd*.

PcmsConnect - Connect to Dimensions Database

Purpose

This function provides you with a connection to the Dimensions database (for example, *intermediate*) specified by the input parameters. You can use this function to open multiple connections on the same or different Dimensions databases. This function will return a *connectId* (integer) that represents your database connection. The database parameters reflect the same values as you would specify for a PCMSDB symbol.

Prototype

```
int
PcmsConnect (
       char           *database,
       char           *password,
       char           *node
);
```

Parameters

database	is the name of the Dimensions database to connect to.
password	is the password of the database. Use NULL if the user is secure
node	is the ORACLE Service Name assigned to the node where the Oracle database is located.

Return Codes

PcmsConnect() returns:

int connectId	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo, PcmsErrorStr, PcmsDbErrorNo,* and *PcmsDbErrorStr.*

Comments

This function will not return until a successful connection has been made, or an error is encountered while attempting the connection.

If you wish to use the default Dimensions database for this user (i.e. PCMSDB), then invoke *PcmsConnect()* with NULL parameters for database, password and node.

This function will apply all the same pre-login user-verification checks as if the user had typed '*pcms*' at the command prompt.

Sample

```
/*
 * connect to Dimensions
 *
 */
int connect()
{
    int conId = PCMS_ERROR
    /* CONNECT to PCMSDB */
    conId = PcmsConnect(NULL, NULL, NULL);
    return conId;
}
```

Related Function

PcmsDisconnect().

PcmsDisconnect - Disconnect from a Dimensions Database

Purpose

This function disconnects from the Dimensions database as specified by the *connectId.* This *connectId* must be a valid *connectId* returned by *PcmsConnect().*

Prototype

```
int
PcmsDisconnect (
       int            connectId
);
```

Parameters

connectId	is the database connection identifier.

Return Codes

PcmsDisconnect() returns:

PCMS_OK	on success
PCMS_ERROR	on failure, and sets *PcmsErrorNo, PcmsErrorStr, PcmsDbErrorNo,* and *PcmsDbErrorStr.*

Sample

```
/* Disconnect and exit with success… */
(void) PcmsDisconnect(conID);
return (EXIT_SUCCESS);
```

Comments

On some operating systems if you do not call this function before the application exits, the connection to the repository may never terminate.

This function does not return until the disconnection is complete.

Related Function

PcmsConnect().

PcmsExecCommand - Execute Dimensions Command Synchronously

Purpose

This function sends a command to Dimensions and waits for it to complete. See the related document "PVCS® Dimensions™ - Command-Line Reference Guide" for information on legal syntax for command mode applications.

Prototype

```
int
PcmsExecCommand (
       int            connectId,
       char           *command,
       char           **response
);
```

Parameters

connectId	is the database connection identifier.
command	is the command to be executed.
response	is the address of a *char** variable that will be set to point to a dynamically allocated buffer containing the diagnostic messages generated during the execution of the command. It is the responsibility of the calling application to free this buffer when it is no longer required.

Return Codes

PcmsExecCommand () returns:

PCMS_OK	on success
PCMS_FAIL	on the Dimensions command failing and sets the *response* parameter.
PCMS_ERROR	on other failures and sets *PcmsErrorNo* and *PcmsErrorStr*

Sample

See the examples on the release media.

Comments

All commands that have been queued previously to Dimensions using *PcmsSendCommand()* will be processed first. When the results from all those commands have been received, the current command will be executed by Dimensions. Therefore the time taken for *PcmsExecCommand()* to process a single simple command may depend on the number of commands previously queued to Dimensions using *PcmsSendCommand()*.

Related Function

PcmsSendCommand().

PcmsSetCallback - Set Dimensions API Server Callback

Purpose

This function sets up a callback function for the specified *connectId*. The previous callback function and associated *clientData* are returned in the *ptrOldPcmsCallback* pointer. The callback function is used to register the results of Dimensions commands submitted asynchronously by the function *PcmsSendCommand()*. The callback function will be invoked when a user calls *PcmsCheckMessages()*.

Prototype

```
int
PcmsSetCallback (
       int                  connectId,
       PcmsCallbackStruct   *ptrNewPcmsCallback,
       PcmsCallbackStruct   *ptrOldPcmsCallback
);
```

Parameters

connectId is the database connection identifier.

ptrNewPcmsCallback is a pointer to a structure of type *PcmsCallbackStruct*. In this structure the member field *callback* is a pointer to the callback function. This function must have the following prototype.

```
void sampleCallbackProc(
        int connectId,
        void *clientData,
        int commandStatus,
        int commandId,
        char *commandStr,
        char *callData,
        ...
)
```

continued

	The *clientData* field of the *PcmsCallbackStruct* will be passed as one of the parameters to the callback function. Specifying a NULL *ptrNewPcmsCallback* parameter installs the default callback for the connection. This is a null function.
ptrOldPcmsCallback	is a pointer to a structure of type *PcmsCallbackStruct*, which will hold the previous callback function and client data. If this information is of no interest then you may specify NULL.

Return Codes

PcmsSetCallback () returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr.*

Comments

The parameters passed to the callback function are:

connectId	is the database connection identifier.
clientData	is the pointer value specified in the *PcmsCallbackStruct* when the callback function was installed using *PcmsSetCallback()*.
commandStatus	is the status of the Dimensions command.
commandId	is the unique command identifier associated with the command. This value corresponds to that returned by *PcmsSendCommand()*.

continued

commandStr	is the text of the command submitted.
callData	is the text output that has resulted from the command execution.
...	is a variable argument list which is used internally by Dimensions. You must not attempt to use this list.

Related Functions

PcmsSendCommand(), *PcmsCheckMessages()*, *PcmsSetNoErrorCallback()*.

PcmsSetDbErrorCallback - Set Server Error Callback

Purpose

This function sets up a callback function to be executed when ORACLE is no longer available. The callback function will be invoked when an application invokes an DTK function and the DTK detects that the Dimensions repository is no longer available. For example, this error may occur if the Dimensions server has been powered down.

Prototype

```
int
PcmsSetDbErrorCallback (
        int                     connectId,
        PcmsCallbackStruct      *ptrPcmsCallback
);
```

Parameters

connectId is the database connection identifier.

ptrPcmsCallback is a pointer to a structure of type *PcmsCallbackStruct.* In this structure the member field *callback* is a pointer to the callback function. This function must have the following prototype.

```
void sampleCallbackProc(
        intconnectId,
        void*clientData,
        intcommandStatus,
        intcommandId,
        char*commandStr,
        char*callData,
        ...
)
```

The *clientData* field of the *PcmsCallbackStruct* will be passed as one of the parameters to the callback function.

Return Codes

PcmsSetDbErrorCallback () returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr.*

Comments

The parameters passed to the callback function are:

connectId	Database connection identifier associated with the command.
clientData	Null
commandStatus	The Oracle error code detected.
commandId	Zero
commandStr	Null
callData	The text output formatted to include the ORACLE error code(s) that has resulted from the command execution.
...	is a variable argument list which is used internally by Dimensions. You must not attempt to use this list.

Related Functions

PcmsSetCallback().

PcmsSendCommand - Execute Dimensions Command Asynchronously

Purpose

This function sends a command to Dimensions and returns without waiting for it to complete. See the related document "PVCS® Dimensions™ - Command-Line Reference Guide" for information on legal syntax for command mode applications.

Prototype

```
int
PcmsSendCommand (
        int             connectId,
        char            *command,
        int             *cmdId
);
```

Parameters

connectId	is the database connection identifier.
command	is the command to execute.
cmdId	is a unique command identifier that is returned to the user.

Return Codes

PcmsSendCommand () returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr*.

Comments

You must call the function *PcmsCheckMessages()* to check whether the results of commands submitted using *PcmsSendCommand()* are available. If results are available, your callback function will be invoked (see *PcmsSetCallback on* page 54).

NOTE If your application does not call *PcmsCheckMessages()* periodically, commands sent with *PcmsSendCommand()* may not be executed by Dimensions.

The commands "EXIT" and "exit" will result in an error being returned. A null string will also result in an error. Use *PcmsDisconnect()* to terminate the connection with the Dimensions Server.

Related Functions

PcmsSetCallback(), *PcmsCheckMessages()*, *PcmsExecCommand()*.

PcmsGetConnectDesc - Get Input File Descriptor

Purpose

This function returns the input file descriptor for the specified connection identifier. The purpose of this function is for an X based application to add the file descriptor as an input for the X application using *XtAddInput()*. When the X application receives notification that there is input available on the file descriptor, the application should call *PcmsCheckMessages().* This will then activate any callback functions that have been setup by *PcmsSetCallback().* Using this method of processing Dimensions messages, the need for *PcmsSetIdleChecker()* is eliminated.

Prototype

```
int
PcmsGetConnectDesc (
        int     connectId
);
```

Parameters

connectId	is the database connection identifier.

Return Codes

PcmsGetConnectDesc () returns:

int fd	on success
PCMS_ERROR	on failure and sets *PcmsErrno* and *PcmsErrorStr*

Comments

This function is applicable only to UNIX.

Related Functions

PcmsCheckMessages(), *PcmsSetCallback()*.

PcmsCheckMessages - Check Results of Dimensions Command

Purpose

This function may be used to check whether the results from Dimensions commands previously submitted, using *PcmsSendCommand()*, are available.

Prototype

```
int
PcmsCheckMessages (
        int             connectId,
        int             flag
);
```

Parameters

connectId	is the database connection identifier.
flag	is used to determine whether the operation is to be blocking (PCMS_MSG_WAIT) or non-blocking (PCMS_MSG_NOWAIT).

Return Codes

PcmsCheckMessages() returns:

PCMS_OK	results of a command are available and the callback function has been invoked
PCMS_FAIL	results of a command are not available (non-blocking operation)
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr*.

Comments

If there are no commands being processed on this connection, PCMS_FAIL will be returned. If there are outstanding commands, the operation of this function will depend on the value of *flag.*

- If *flag* is equal to PCMS_MSG_WAIT, the function will block (by calling the function set with *PcmsSetIdleChecker()*) until the results of the next command are available. The function will then invoke the callback function (see *PcmsSetCallback()* on page 54) and return PCMS_OK.
- If *flag* equals PCMS_MSG_NOWAIT, the function will return immediately if no results are available (after calling the function set with *PcmsSetCallback()*), and the return value will be PCMS_FAIL. If results are available, the callback function will be invoked and the value PCMS_OK returned.

Related Functions

PcmsSendCommand(), *PcmsSetCallback()*, *PcmsSetIdleChecker()*.

PcmsSetDirectory - Change Dimensions Default Directory

Purpose

This function changes the default directory of the Dimensions process being managed by your application.

Prototype

```
int
PcmsSetDirectory (
        int             connectId,
        char            *new_directory
);
```

Parameters

connectId	is the database connection identifier.
new_directory	is the full specification of the default directory the Dimensions process is to use.

Return Codes

PcmsSetDirectory() returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr*.

Comments

The function does not change directory until all commands in the *PcmsSendCommand()* queue (if any) are executed.

The directory change effects this connection only.

Related Functions

PcmsSendCommand().

PcmsGetWsetObj - Get User's Current Workset

Purpose

This function returns the current workset in which this Dimensions session is active.

Prototype

```
int
PcmsGetWsetObj (
        int             connectId,
        int             options,
        PcmsObjStruct   **ptrPcmsObjStruct);
```

Parameters

connectId	is the database connection identifier.
options	is not currently supported (use 0).
ptrPcmsObjStruct	is the address of the workset object that points to a dynamically allocated buffer containing a *PcmsObjStruct*, whose *objType* field will be PCMS_WORKSET.

Return Codes

PcmsGetWsetObj() returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

Sample

```
static char*
GetWorkset(int conId)
{
    static char     workset_id[PCMS_L_PRODUCT_ID + PCMS_L_CD_ID + 5];
    PcmsObjStruct   *wsobj = (PcmsObjStruct *)0;
    workset id[0] = '\0';
    switch(PcmsGetWsetObj(conId, 0, &wsobj))
    {
        case    PCMS ERROR:
        case    PCMS FAIL:
                return((char *)0);
        default:
            break;
    }
    (void)sprintf(workset_id,"\"%s\":\ "%s\"",
        wsobj->productId,wsobj->objId);
    return(worksetid);
}
```

PcmsSetWsetObj - Set User's Current Workset

Purpose

This function allows the user to reset the current workset in which this Dimensions session is active.

Prototype

```
int
PcmsSetWsetObj (
        int     connectId,
        int     options,
        PcmsObjStruct  *ptrPcmsObjStruct,
        char    *dir);
```

Parameters

connectId	is the database connection identifier.
options	is not currently supported (use 0).
ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* that is populated by the user to indicate what workset to change to.
dir	is the workset directory to use.

Return Codes

PcmsSetWsetObj() returns:

PCMS_OK	on success
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

PcmsObjGetRels - Get Dimensions Object Relationships

Purpose

This function can be used to navigate objects and their relationships to other objects. For example, to return successor revisions of an item.

Prototype

```
int
PcmsObjGetRels (
       int            connectId,
       int            fromObjUid,
       int            objType,
       int            options,
       int            contextUid,
       int            *noRels,
       PcmsRelStruct  **ptrPcmsRelStruct);
```

Parameters

connectId	is the database connection identifier.
fromObjUid	is the integer uid for a Dimensions object.
objType	is the type of the Dimensions object. This type must be one of PCMS_PART, PCMS_ITEM, PCMS_CHDOC, or PCMS_BASELINE.
options	is a collection of bits set that indicates the type of objects to return in *ptrPcmsRelStruct*. If this value is zero, then all object relationship types are returned. You can restrict the list of objects returned by specifying one or more of the following types PCMS_PART, PCMS_ITEM, PCMS_CHDOC, or PCMS_BASELINE.

continued

contextUid	can be used to limit the objects navigated to a specific baseline_uid.
noRels	is a pointer to an integer variable in which to store the number of structures returned in *ptrPcmsRelStruct*.
ptrPcmsRelStruct	is a pointer to a contiguous block of allocated memory that contains a number of structures of type *PcmsRelStruct*. If no objects are found then *noRels* is set to zero and *ptrPcmsRelStruct* is set to (*PcmsRelStruct* *) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsObjGetRels() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects were found
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

Comments

If *fromObjUid* is zero, then the top of the Dimensions database is used as the starting point for the navigation.

If *fromObjUid* is zero and *options* is set to PCMS_PART, then only the Dimensions products will be returned.

If PCMS_OPT_LATEST is set, then only the latest version of any related objects is returned. This option is only valid when *objType* is set to either PCMS_ITEM or PCMS_PART, and only applies to 'item to design part' relationships.

If PCMS_OPT_SUCC is set, then *contextUid* will be ignored and the *fromObjUid*'s successor revision will be returned. This option is only valid when *objType* is set to either PCMS_PART or PCMS_ITEM.

The option PCMS_OPT_MERGED can be used in conjunction with PCMS_OPT_SUCC to return objects that have resulted as a merge based on the object specified in *fromObjUid.* Currently this option is only available for items.

The following table lists the combinations of object references, object types and query options that are valid for this function. Note that the PCMS_BASELINE option is always invalid when obtaining the relationships recorded in a baseline.

fromObjUid	objType	Options			
		PART	ITEM	BASELINE	CHDOC
Zero	Ignored	V	V	V	V
Non-zero	PART	V	V	I	I
Non-zero	ITEM	I	V	I	I
Non-zero	BASELINE	V	V	I	I
Non-zero	CHDOC	V	V	I	V

Key: V = Valid I = Invalid

Related Functions

PcmsGetBackRels().

PcmsObjGetBackRels - Get Dimensions Object Reverse Relationships

Purpose

This function can be used to navigate objects and their relationships to other objects. For example, to return predecessor revisions of an item. This function performs the inverse navigation of *PcmsObjGetRels()*.

Prototype

```
int
PcmsObjGetBackRels (
       int            connectId,
       int            fromObjUid,
       int            objType,
       int            options,
       int            contextUid,
       int            *noRels,
       PcmsRelStruct  **ptrPcmsRelStruct);
```

Parameters

connectId	is the database connection identifier.
fromObjUid	is the integer uid for a Dimensions object.
objType	is the type of the Dimensions object. This type must be one of PCMS_PART, PCMS_ITEM, PCMS_CHDOC, or PCMS_BASELINE.

continued

options	is a collection of bits set that indicates the type of objects to return in *ptrPcmsRelStruct*. If this value is zero, then all object relationship types are returned. You can restrict the list of objects returned by specifying one or more of the types PCMS_PART, PCMS_ITEM, PCMS_CHDOC, or PCMS_BASELINE.
contextUid	can be used to limit the objects navigated to a specific baseline_uid.
noRels	is a pointer to an integer variable in which to store the number of structures returned in *ptrPcmsRelStruct*.
ptrPcmsRelStruct	is a pointer to a contiguous block of allocated memory that contains a number of structures of type *PcmsRelStruct*. If no objects are found then *noRels* is set to zero and *ptrPcmsRelStruct* is set to (*PcmsRelStruct* *) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsObjGetBackRels() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects where found
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Comments

If PCMS_OPT_PRED is set, then *contextUid* will be ignored and the *fromObjUid*'s predecessor revision will be returned. This

option is only valid when *objType* is set to either PCMS_PART or PCMS_ITEM.

The value of *fromObjUid* cannot be specified as zero.

The option PCMS_OPT_MERGED can be used in conjunction with PCMS_OPT_PRED to return objects that have been used in a merge to create the object specified in *fromObjUid.* Currently this option is only available for items.

The following table lists combinations of object reference, object type and query options that are valid for this function. Note that the PCMS_BASELINE option is always invalid when obtaining the reverse relationships recorded in a baseline.

objType	Options			
	PART	ITEM	B'LINE	CHDOC
PART	V	I	V	V
ITEM	V	V	V	V
BASELINE	I	I	I	I
CHDOC	I	I	I	V

Key: V = Valid I = Invalid

Related Functions

PcmsObjGetRels().

PcmsQuery - Find Dimensions Objects, returning Uids

Purpose

This function finds a list of Dimensions objects from the fields specified in the *PcmsObjStruct*. If values are present in the fields of the *ptrPcmsObjStruct,* they will be used to further refine the query. The only field in *ptrPcmsObjStruct* that must be filled in is *objType*. This means that the returned object uids will all be of the same type.

Prototype

```
int
PcmsQuery (
       int                  connectId,
       PcmsObjStruct        *ptrPcmsObjStruct,
       int                  options,
       int                  *noObjs,
       int                  **ptrObjUids
);
```

Parameters

connectId	is the database connection identifier.
PtrPcmsObjStruct	is a pointer to a *PcmsObjStruct* that will be used to further refine the query. The *objType* field in the *PcmsObjStruct* must be one of PCMS_ITEM, PCMS_PART, PCMS_CHDOC, PCMS_USER, PCMS_BASELINE, or PCMS_WORKSET
options	is a collection of bits that is used to change the default behavior of this function.

continued

noObjs	is a pointer to an integer variable in which to store the number of integer uids returned in *ptrObjUids.*
ptrObjUids	is a pointer to a contiguous block of allocated memory that lists the uids that the query returned. If no objects are found as a result of the function call, then *noObjs* is set to zero and *ptrObjUids* is set to (int *) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsQuery() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects were found
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Sample

```
/*
*-------------------------------------------------------------
* FUNCTION SPECIFICATION
* Name:
*  CountPcmsItems
* Description:
*  Count items in Dimensions
* Parameters:
*  char *itemId
* Return:
*  no of items
* Notes:
*-------------------------------------------------------------
*/
```

continued

```
static int
CountPcmsItems(int conId, char *itemId)
{
    /*
     * Query the database for items with the itemId passed in..
     */
    int *uids = 0;
    int noUids = 0;
    int noAttrs = 2;
    int i = 0;
    int xx = PCMS_OK;
    PcmsObjStruct obj = { 0 };
    #define SET_ATTR(_attr,_value,p) \
    {\
        register int x = p;\
        x--;\
        obj.attrs[x].attr = _attr;\
        PcmsSvaSetVal(obj.attrs[x].value,_value,0);\
        p++;\
    }
    obj.objType = PCMS_ITEM;
    obj.noAttrs = noAttrs;
    obj.attrs = 0;
    obj.attrs =
        (PcmsObjAttrStruct *)
        PcmsEvntCalloc(sizeof(PcmsObjAttrStruct)
        *obj.noAttrs);
    (void)strcpy(obj.objId,itemId);
    i=1;
    /* Search for items with TXT as format and of */
    /* SPEC_UID 121 */
    SET_ATTR(PCMS_ATTR_FORMAT,"TXT",i);
    SET_ATTR(PCMS_ATTR_ITEM_SPEC_UID,"121",i);
    /* Run the query */
    if ((xx = PcmsQuery(conId,&obj,0,&noUids,&uids))==PCMS_OK)
    {
        /* Free memory */
        if (uids && noUids > 0)
            PcmsEvntFree((int *)uids);
    }
    else
    {
        (void)fprintf(stdout,"\nNo objects found - %s",
                    (xx == PCMS_ERROR) ? "Error" : "Fail");
        if (xx == PCMS_ERROR)
            (void)fprintf(stdout,PcmsErrorStr);
    }
    /* Free memory */
    (void)PcmsObjFree(&obj);
    return ((xx == PCMS_ERROR) ? xx : noUids);
}
```

Comments

By manipulating the **attrs* pointer and associated *noAttrs* members of the *PcmsObjStruct* structure it is possible to use system and user attributes as additional components within the query. If you wish to make use of this functionality, then you only need to specify values in the *attr* and *value* members of the *PcmsObjAttrStruct* structure. All other member fields are ignored.

Both the members of the *PcmsObjStruct* structure and the *value* field member of the *PcmsObjAttrStruct* structure support the use of wildcard characters. There are two different kinds of wildcard that you can use:

- ' %' (per cent) which allows pattern matching on many characters
- ' _ ' (underscore) which allows pattern matching against one character.

If the *objType* field member of the *PcmsObjStruct* is set to PCMS_CHDOC, you can use the *options* parameter (PCMS_OPT_SECONDARY_CATALOGUE) to make the function query against the secondary change document catalog instead of the primary catalog. By default, the function will always query the primary change document catalog.

If the *objType* field member of the *PcmsObjStruct* is set to PCMS_ITEM, you can use the options parameter (PCMS_OPT_LATEST) to return only the *latest* revisions of the items that match the query.

If you use attribute filters or the *options* parameter to further restrict the list of uids returned, the speed of the query will be affected.

Only the following system attributes are supported in this function.

Object	Attribute
PCMS_PART	PCMS_ATTR_PARTNO PCMS_ATTR_LOCALNO
PCMS_ITEM	PCMS_ATTR_FORMAT PCMS_ATTR_FILENAME PCMS_ATTR_ITEM_SPEC_UID PCMS_ATTR_LIB_FILENAME PCMS_ATTR_COMPRESSED PCMS_ATTR_SENDER_ID PCMS_ATTR_CREATE_DATE PCMS_ATTR_ORIGINATOR PCMS_ATTR_PHASE PCMS_ATTR_LIFECYCLE
PCMS_BASELINE	PCMS_ATTR_TEMPLATE PCMS_ATTR_BASELINE_TYPE
PCMS_CHDOC	PCMS_ATTR_CREATE_DATE PCMS_ATTR_ORIGINATOR PCMS_ATTR_PHASE PCMS_ATTR_SUPER_TYPE PCMS_ATTR_UPDATE_DATE PCMS_ATTR_LIFECYCLE
PCMS_USER	PCMS_ATTR_GROUP PCMS_ATTR_FULL_USERNAME PCMS_ATTR_PHONE PCMS_ATTR_DEPT PCMS_ATTR_SITE
PCMS_WORKSET	PCMS_ATTR_TRUNK (PCMS_ATTR_TRUNC) PCMS_ATTR_ENFORCE_REV

If you use a multi-valued attribute as a filter, only the first element of the attribute list will be used. The other elements will be ignored.

Related Functions

PcmsFullQuery().

PcmsObjInSecondary - Is Change Document Object in Secondary Catalog

Purpose

This macro will return an integer indicating whether the object specified by the parameter *objPtr* is a Change Document in the secondary catalog.

Prototype

```
int
PcmsObjInSecondary (
        PcmsObjStruct          *objPtr
);
```

Parameters

objPtr	is a pointer to a *PcmsObjStruct*.

Return Codes

PcmsObjInSecondary() returns:

PCMS_OK	if this change document is in the secondary catalog
PCMS_FAIL	if this change document is in the primary catalog.

PcmsFullQuery - Find Dimensions Objects, returning Complete Objects

Purpose

This function, like *PcmsQuery()*, will return a set of Dimensions objects based on a user-specified filter. However, unlike *PcmsQuery(),* this function returns fully populated *PcmsObjStruct*s with both object and attribute details loaded. This function is faster than *PcmsQuery()* for returning large amounts of data.

Prototype

```
int
PcmsFullQuery (
       int                  connectId,
       PcmsObjStruct        *ptrPcmsObjStruct,
       int                  options,
       int                  *noObjs,
       PcmsObjStruct        **ptrObjs
);
```

Parameters

connectId	is the database connection identifier.
ptrPcmsObjStruct	is a pointer to a *PcmsObjStruct* that contains the fields to query for. The *objType* field in the *PcmsObjStruct* must currently only be PCMS_CHDOC.
options	if this is set to PCMS_OPT_SECONDARY_CATALOGUE, then the function will process change documents in the secondary catalog. By default this function processes change documents in the primary catalog.

noObjs	is a pointer to an *PcmsObjStruct* variable in which to store the objects returned in *ptrObjs.*
ptrObjs	is a pointer to a contiguous block of allocated memory that contains the structures that the query returned. If no objects are found *noObjs* is set to zero and *ptrObjs* is set to (*PcmsObjStruct* *) zero. It is the responsibility of the calling application to free this memory when it is no longer required.

Return Codes

PcmsFullQuery() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects were found
PCMS_ERROR	on failure and sets *PcmsErrorNo, PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr*.

Sample

```
 /*
 *-------------------------------------------------------------
 * FUNCTION SPECIFICATION
 * Name:
 *  CountPcmsChdocs
 * Description:
 *  Count chdocs
 * Return:
 * Return no of chdocs
 *-------------------------------------------------------------
 */
static int
CountPcmsChdocs(int conId, char *chdoc)
{
    /*
     * Query the database for chdocs with the chdoc passed in...
     */
    int *uids = 0;
```

continued

```
    int noUids = 0;
    int noAttrs = 5;
    int i = 0;
    int xx = PCMS_OK;
    PcmsObjStruct obj = { 0 };
    PcmsObjStruct *ptrObjs = 0;
    #define SET_ATTR(_attr,_value,p) \
    {\
        register int x = p;\
        x--;\
        obj.attrs[x].attr = _attr;\
        PcmsSvaSetVal(obj.attrs[x].value,_value,0);\
        p++;\
    }
    obj.objType = PCMS_CHDOC;
    obj.noAttrs = noAttrs;
    obj.attrs = 0;
    obj.attrs =
        (PcmsObjAttrStruct *)
         PcmsEvntCalloc(sizeof(PcmsObjAttrStruct)
            * obj.noAttrs);
    (void)strcpy(obj.objId,chdoc);
    i=1;
    /* Search for chdocs with fixed attributes */
    SET_ATTR(PCMS_ATTR_CREATE_DATE,"%%",i);
    SET_ATTR(PCMS_ATTR_ORIGINATOR,"%%",i);
    SET_ATTR(PCMS_ATTR_PHASE,"%%",i);
    SET_ATTR(PCMS_ATTR_SUPER_TYPE,"%%",i);
    SET_ATTR(PCMS_ATTR_UPDATE_DATE,"%%",i);
    if ((xx = PcmsFullQuery(conId,&obj,0,
                &noUids,
                &ptrObjs))==PCMS_OK)
    {
         /* Free memory */
         if (ptrObjs && noUids > 0)
         {
             int xc = 0;
             for(xc=0;xc<noUids;xc++)
                 PcmsObjFree(&ptrObjs[xc]);
         }
    }
    else
    {
         (void)fprintf(stdout,"\nNo objects found - %s",
                (xx == PCMS_ERROR) ? "Error" : "Fail");
         if (xx == PCMS_ERROR)
             (void)fprintf(stdout,PcmsErrorStr);
     }
     /* Free memory */
     (void)PcmsObjFree(&obj);
     return ((xx == PCMS_ERROR) ? xx : noUids);
}
```

Comments

By manipulating the **attrs* pointer and associated *noAttrs m*embers of the *PcmsObjStruct* structure it is possible to use system and user attributes as additional components within the query. If you wish to make use of this functionality, then you only need to specify values in the *attr* and *value* members of the *PcmsObjAttrStruct* structure. All other member fields are ignored.

Both the members of the *PcmsObjStruct* structure and the *value* field member of the *PcmsObjAttrStruct* support the use of wildcard characters. There are two different kinds of wildcard that you can use:

- ' % ' (per cent) which allows pattern matching on many characters
- ' _ ' (underscore) which allows pattern matching against one character.

This function currently supports only objects of PCMS_CHDOC. **It is liable for change in the future**.

You can only use the following as system attribute filters:

- PCMS_ATTR_CREATE_DATE
- PCMS_ATTR_ATTR_ORIGINATOR
- PCMS_ATTR_PHASE
- PCMS_ATTR_SUPER_TYPE
- PCMS_ATTR_UPDATE_DATE
- PCMS_ATTR_LIFECYCLE.

If you use a multi-valued attribute as a filter, then only the first element of the attribute list will be used, the other elements will be ignored.

Related Functions

PcmsQuery().

PcmsPendGet - Retrieve Dimensions Objects Pending for a User

Purpose

This function retrieves the pending list of items and/or change documents for the current or a specified user.

CAUTION! The parameter *userName* will turn into an object uid for a Dimensions user object in the future.

Prototype

```
int
PcmsPendGet (
       int                  connectId,
       char                 *userName,
       char                 *reserved,
       int                  options,
       int                  *noStructs,
       PcmsPendStruct       **ptrPcmsPendStructs
);
```

Parameters

connectId	is the database connection identifier.
userName	is the user to query the objects pending for. If this parameter is NULL then the current user's pending list is queried.
reserved	is reserved for future use.

continued

options	this determines which objects are to be returned by this function. You can specify one of the following: ■ Zero (0) which returns all pending object types ■ PCMS_CHDOC which returns all pending change documents ■ PCMS_ITEM which returns all pending items.
noStructs	is a pointer to an integer variable in which to store the number of structures of type *PcmsPendStruct* returned in *ptrPcmsPendStruct*.
ptrPcmsPendStructs	is a pointer to a contiguous block of allocated memory that lists the objects that the function returned. If no objects are found, then this value is set to 0 and *ptrPcmsPendStructs* is set to (*PcmsPendStruct**) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsPendGet() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects are found
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Comments

Only if you have the role of CHANGE-MANAGER can you use the *userName* parameter to query another user's change document pending list.

Only if you have the role of PRODUCT-MANAGER can you use the *userName* parameter to query another user's item pending list.

PcmsPendWhoGet - Retrieve Users for Object

Purpose

This function retrieves the users who will have a specified object pending for them at a user-defined status.

Prototype

```
int
PcmsPendWhoGet (
       int                  connectId,
       int                  objUid,
       int                  objType,
       int                  options,
       char                 *status,
       int                  *noStructs,
       PcmsPendStruct       **ptrPcmsPendStructs
);
```

Parameters

connectId	is the database connection identifier.
objUid	is the uid for a Dimensions object against which this function will be run.
	NOTE *objUid* cannot be zero (0).
objType	is the type of the Dimensions object. This type must be one of PCMS_PART, PCMS_ITEM, or PCMS_CHDOC.
status	is the status in the lifecycle for which you wish to return the list of pending users.
options	is reserved for future use.

continued

noStructs	is a pointer to an integer variable in which to store the number of structures of type *PcmsPendStruct* returned in *ptrPcmsPendStructs*.
ptrPcmsPendStructs	is a pointer to a contiguous block of allocated memory that lists the users that the function has found. If no objects are found *noStructs* is set to zero and *ptrPcmsPendStructs* is set to (*PcmsPendStructs* *) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsPendWhoGet() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects were found
PCMS_ERROR	on failure and sets *PcmsErrorNo, PcmsErrorStr, PcmsDbErrorNo,* and *PcmsDbErrorStr*

Related Functions

PcmsPendGet().

PcmsCntrlPlanGet - Get Dimensions Process Model Information

Purpose

This function queries the Dimensions process model (control plan) and returns the data in various different structures.

Prototype

```
int
PcmsCntrlPlanGet (
        int             connectId,
        int             reserved,
        int             options,
        char            *fromId,
        int             objUid,
        char            *startContext,
        int             *noStructs,
        void            **ptrStructs
);
```

Parameters

When *options* = PCMS_CHD_TYPE:

objUid	is 0 for all change document types or contains a *typeUid* from a *PcmsObjStruct* structure.
fromId	is the Dimensions product from which to retrieve the change document types when *objUid* is 0. This parameter is ignored when *objUid* is not 0.
startContext	is NULL or a valid Dimensions super_type cast to a char *. This parameter is ignored when *objUid* is not 0.
ptrStructs	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsTypeStruct*.

When *options* = PCMS_PART_TYPE or PCMS_PART_CATEGORY:

objUid	is 0 for all part types or contains a *typeUid* from a *PcmsObjStruct* structure.
fromId	is the Dimensions product from which to retrieve the design part types when *objUid* is 0. This parameter is ignored when *objUid* is not 0.
startContext	is ignored.
ptrStructs	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsTypeStruct*.

When *options* = PCMS_BASELINE_TYPE:

objUid	is 0 for all baseline types or contains a *typeUid* from *PcmsObjStruct* structure.
fromId	is the Dimensions product from which to retrieve the baseline types when *objUid* is 0. This parameter is ignored when *objUid* is not 0.
startContext	is ignored.
ptrStructs	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsTypeStruct.*

When *options* = PCMS_LC:

fromId	is the lifecycle-id.
objUid	is ignored.
startContext	is NULL (in which case the first lifecycle state is returned) or is a valid state within the lifecycle (in which case the next possible states will be returned).
ptrStructs	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsLcStruct*.

When *options* = PCMS_ATTRIBUTE ORed with PCMS_OBJ_TYPE ORed with PCMS_ITEM or PCMS_PART or PCMS_CHDOC or PCMS_USER or PCMS_BASELINE:

fromId	is the Dimensions product from which to retrieve the attribute definitions when *objUid* is 0. This parameter is ignored when *objUid* is not 0.
objUid	is 0 or the uid of the type for which to retrieve the attribute definitions.
startContext	is NULL or a valid name of a Dimensions type (the *typeName* field of the *PcmsTypeStruct*). If a *startContext* is specified, then only attribute definitions that have been specified in the documentation plan will be returned. This parameter is ignored when *objUid* is not 0.
ptrStructs	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsObjAttrDefStruct.*

The following parameters apply whatever the definition of *options.*

connectId	is the database connection identifier.
reserved	is reserved for future use.
noStructs	is a pointer to an integer variable in which to store the number of structures returned in *ptrStructs*. If no objects are found *noStructs* is set to zero and *ptrStructs* is set to (void *) zero.

It is the responsibility of the calling application to free the *ptrStructs* pointer when it is no longer required.

Return Codes

PcmsCntrlPlanGet() returns:

PCMS_OK	on success
PCMS_FAIL	when no objects were found
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

PcmsInitSpec - Get Dimensions Object Details by Specification

Purpose

This function populates a *PcmsObjStruct* with the details on a specific object.

Prototype

```
int
PcmsInitSpec (
        int                     connectId,
        char                    *objSpec,
        int                     objType,
        PcmsObjStruct           *ptrPcmsObjStruct
);
```

Parameters

connectId	is the database connection identifier.
objSpec	is the textual specification of an object e.g. "*FS:HITOMI_C.A-SRC;main#1*".
objType	is the type of the object that the specification refers to. You can use PCMS_PART, PCMS_ITEM, PCMS_BASELINE, PCMS_USER, PCMS_CHDOC or PCMS_WORKSET.
ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* in which to store the details on the object specified by *objSpec.*

Return Codes

PcmsInitSpec() returns:

PCMS_OK	on success
PCMS_FAIL	on not finding the object
PCMS_ERROR	on failure and sets *PcmsErrorNo, PcmsErrorStr, PcmsDbErrorNo,* and *PcmsDbErrorStr*.

Comments

This function does not populate the attribute details within the *PcmsObjStruct* structure. This has to be done separately by calling *PcmsGetAttrs().*

Related Functions

PcmsInitUid(), *PcmsGetAttrs()*.

PcmsInitUid - Get Dimensions Object Details by Uid

Purpose

This function populates a *PcmsObjStruct* with the details on a specific object.

Prototype

```
int
PcmsInitUid (
        int                     connectId,
        int                     objUid,
        int                     objType,
        PcmsObjStruct           *ptrPcmsObjStruct
);
```

Parameters

connectId	is the database connection identifier.
objUid	is the integer uid of the object.
objType	is the type of the object that the specification refers to. You can use PCMS_PART, PCMS_ITEM, PCMS_BASELINE, PCMS_USER, PCMS_CHDOC or PCMS_WORKSET.
ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* in which to store the details on the object specified by the uid.

Return Codes

PcmsInitUid() returns:

PCMS_OK	on success
PCMS_FAIL	on not finding the object
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Comments

This function does not populate the attribute details within the *PcmsObjStruct* structure. This has to be done separately by calling *PcmsGetAttrs().*

Related Functions

PcmsInitSpec(), *PcmsGetAttrs()*.

PcmsSetAttrs - Set Dimensions Object Attributes

Purpose

This function uses the attribute details defined in a *PcmsObjStruct* and sets these attributes on the appropriate Dimensions object.

By using this function you can populate a *PcmsObjStruct* with the details on an object, such as an item, manipulate the *noAttrs* and **attrs* member fields and then apply these attributes to the Dimensions object.

Prototype

```
int
PcmsSetAttrs (
        int                     connectId,
        PcmsObjStruct           *ptrPcmsObjStruct
);
```

Parameters

connectId	is the database connection identifier.
ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* that has been initialized via *PcmsInitUid()* or *PcmsInitSpec()* and into which the new attributes have been defined. Each element of the array pointed to by the **attrs* field must have the *attr* and *value* member fields set.

Return Codes

PcmsSetAttrs() returns:

PCMS_OK	on success
PCMS_FAIL	on not setting the attributes successfully. *PcmsErrorStr* will contain the message returned from Dimensions
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr*.

Comments

PcmsSetAttrs() can set user-defined attributes on parts, items, change documents but non-visible attributes (those which are not displayed in client interfaces) can be set only on change documents.

The **attrs* pointer in the *PcmsObjStruct* must only be populated with attributes that you wish to add or modify.

This function can be used only to setup user-defined attributes on objects of type PCMS_ITEM, PCMS_CHDOC and PCMS_PART.

This function enforces the same checks used when setting attribute values as performed by any other interface.

This function is intended for applications using the *Client Architecture Model*. It is not supported when called from within DTK events. If you wish to change attributes from within an event, then please use the Validate event as described in Chapter 5.

Related Functions

PcmsInitUid(), PcmsInitSpec(), PcmsGetAttrs().

PcmsGetAttrs - Get Dimensions Object Attributes

Purpose

This function populates a specific *PcmsObjStruct* with the attribute details for that object. Calling this function will result in the *noAttrs* and **attrs* member elements being populated with the full attribute details and attribute definitions. If you wish to access the information on the attribute definitions, use the *PcmsObjAttrDefStruct* pointer (*attrDef*) from the *PcmsObjAttrStruct* (*attrs*) pointer.

Prototype

```
int
PcmsGetAttrs (
        int                     connectId,
        PcmsObjStruct           *ptrPcmsObjStruct
);
```

Parameters

connectId	is the database connection identifier.
ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* into which the attribute information will be populated.

Return Codes

PcmsGetAttrs() returns:

PCMS_OK	on success
PCMS_FAIL	on not finding the object
PCMS_ERROR	on failure and sets *PcmsErrorStr*, *PcmsDbErrorNo,* and *PcmsDbErrorStr*.

Comments

Before calling this function ensure that you have populated a valid *PcmsObjStruct* via the *PcmsInitUid()* or *PcmsInitSpec()* functions.

When you have populated an object structure using this function, remember to free the memory associated with it via *PcmsObjFree()* when that object is no longer required.

Related Functions

PcmsInitUid(), PcmsInitSpec(), PcmsObjFree()

PcmsObjFree - Free Dimensions Object Structures

Purpose

This function frees any memory that may have been allocated internally to the *PcmsObjStruct* structure. This includes any attributes or attribute definition structures.

Prototype

```
int
PcmsObjFree (
        PcmsObjStruct          *ptrPcmsObjStruct
);
```

Parameters

ptrPcmsObjStruct	is a pointer to a structure of type *PcmsObjStruct* from which to free the memory.

Return Codes

PcmsObjFree() returns:

PCMS_OK	this value is always returned.

PcmsGetAttrDefNum - Get Attribute Definition Number

Purpose

This function returns the attribute number for a specified attribute definition.

Prototype

```
int
PcmsGetAttrDefNum (
       int            connectId,
       char           *productId,
       int            objType,
       char           *attrName,
       int            *attrNum
);
```

Parameters

connectId	is the database connection identifier.
productId	is the product name
objtype	is the type of the object. You can use PCMS_PART, PCMS_ITEM, PCMS_BASELINE, PCMS_USER or PCMS_CHDOC.
attrName	is the name of the attribute (the *variable* field in the *PcmsObjAttrDefStruct*).
attrNum	is the returned attribute number.

Return Codes

PcmsGetAttrDefNum() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find the specified attribute name for the given product and object type
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr.*

PcmsAttrDefInit - Get Attribute Definition

Purpose

This function retrieves an attribute definition for a specified *typeUid*, object type and attribute number.

Prototype

```
int
PcmsAttrDefInit (
       int                    connectId,
       int                    typeUid,
       int                    objType,
       int                    attrNum,
       PcmsObjAttrDefStruct   **ptrDefStruct
);
```

Parameters

connectId	is the database connection identifier.
typeUid	is the type (the *typeUid* field of the *PcmsObjStruct*) to which the attribute applies.
objType	is the type of the object. You can use PCMS_PART, PCMS_ITEM, PCMS_BASELINE, PCMS_USER or PCMS_CHDOC.
attrNum	is the attribute number for which the definition is to be retrieved.
ptrDefStruct	is the address of a pointer to a *PcmsObjAttrDefStruct* in which the attribute definition will be stored. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsAttrDefInit() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find the specified attribute number for the given type and object type
PCMS_ERROR	on failure and sets *PcmsErrorNo* and *PcmsErrorStr.*

Related Functions

PcmsCntrlPlanGet().

PcmsAttrGetLov - Get Attribute's List of Values

Purpose

This function retrieves the list of valid set values that are allowed for a specified object and attribute. The list of values is returned as an array of char *(s).

The *PcmsObjStruct* that is used in this function must have at least the following member fields defined.

- The *typeUid* set to the Dimensions type against which the attribute has been assigned.
- The *noAttrs* field set to at least 1.
- The **attrs* pointer set to a *PcmsObjAttrStruct* structure which must have the member fields set as follows:
  - attr attribute number you are querying
  - value a " " string via *PcmsSvaSetVal()*
  - *attrDef* the corresponding *PcmsObjAttrDefStruct.*

It is possible to obtain the *typeUids* for a product and *objType* via *PcmsCntrlPlanGet()*. A certain attribute definition can then be obtained by calling *PcmsAttrDefInit()*.

Prototype

```
int
PcmsAttrGetLov
 (
       int                  connectId,
       PcmsObjStruct        *objPtr,
       int                  attrNum,
       char                 **message,
       int                  *noStrings,
       char                 ***ptrArrayOfStrings
);
```

Parameters

connectId	is the database connection identifier.
objPtr	is the object containing the type and attribute details.
attrNum	is the attribute number.
message	is the error message returned if the status is not PCMS_OK. It is the caller's responsibility to free this allocated memory if not NULL.
noStrings	is the address of an integer variable to contain the number of strings in the array.
ptrArrayOfStrings	is the address of a pointer to the array of returned strings. This array must be freed via *PcmsLovFree().*

Return Codes

PcmsAttrGetLov() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find the specified attribute number for the given type and object type
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

Sample

```
/*
*-------------------------------------------------------------
*              FUNCTION SPECIFICATION
*              Name:
*               GetLovs
*              Description:
*               Get LOVs for a specified attribute and typeUid
*              Parameters:
*               int  conId
*               int  objType
*               int  typeUid
*               int  attr
*              Return:
*               int
*              Notes:
*-----------------------------------------------------
*/

int GetLovs(int conId, int objType, int typeUid, int attr)
{
    PcmsobjStruct obj = { 0 };
    PcmsObjAttrDefStruct *attrDef = 0;
    char **vals = 0;
    int  noVals = 0;
    int  x = 0;
    char *ptrError = 0;
    int  status = 0;

    obj.typeUid = typeUid;
    obj.noAttrs = 1;
    /* Get the details on the specified */
    /* attribute */
    if ((status = PcmsAttrDefInit(conId,typeUid,
                      objType,attr,&attrDef))!=PCMS_OK)
                      return status;
    /* Put together a dummy object structure */
    obj.attrs =
        (PcmsObjAttrStruct*)PcmsEvntMalloc(sizeof
        (PcmsObjAttrStruct)*1);
    obj.attrs[0].attr = attr;
    PcmsSvaSetVal(obj.attrs[0].value,NULL,0);
    obj.attrs[0].attrDef = attrDef;

    /* Get the LOV values */
    if ((status = PcmsAttrGetLov(conId,&obj,
                      attr,&ptrError,
                      &noVals,&vals))==PCMS_OK)
    {
         int i = 0;
                /**
```

continued

```
                ** Scan list of LOVs and report

         **/
        for(i=0;i<noVals;i++)
           (void)fprintf(stdout,
                  "\nLov[%d/%d] - %s",
                  i,noVals,vals[i]);
        PcmsLovFree(noVals,vals);
    }
    (void)PcmsObjFree(&obj);
    return(status);
}
```

Related Functions

PcmsAttrDefInit(), *PcmsAttrValidate()*.

PcmsAttrValidate - Validate an Attribute Value

Purpose

This function verifies that any attribute values specified on a given object structure do not conflict with any valid sets that may have been defined.

Prototype

```
int
PcmsAttrValidate (
       int                  connectId,
       PcmsObjStruct        *objPtr,
       int                  attrNum,
       char                 **message
);
```

Parameters

connectId	is the database connection identifier.
objPtr	is the object containing the attribute values to be validated.
attrNum	is the attribute number against which the object attributes will be validated.
message	is the error message returned if the status is not PCMS_OK. It is the caller's responsibility to free this allocated memory if not NULL.

Return Codes

PcmsAttrValidate() returns:

PCMS_OK	on success
PCMS_FAIL	on failure
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

Related Functions

PcmsAttrGetLov(), *PcmsLovFree()*.

PcmsLovFree - Free a List of Values

Purpose

This function frees an array of strings returned by *PcmsAttrGetLov()*.

Prototype

```
int
PcmsLovFree (
       int            noValues,
       char           **values
);
```

Parameters

noValues	is the number of strings in the array.
values	is the array name.

Return Codes

PcmsLovFree() returns:

PCMS_OK	on success.

Related Functions

PcmsAttrGetLov().

PcmsGetUserRoles - Obtain User Role Structures

Purpose

This function returns those users who are found to have certain roles for this object. The returned structures indicate whether or not the user was delegated the role (via DLGC), or if the role was inherited from the design tree. These structures will also indicate if the user's capability is primary or secondary.

Currently this function will support only objects of type PCMS_CHDOC. The actionable field of the returned *PcmsUserRoleStruct* is not populated.

Prototype

```
int
PcmsGetUserRoles (
        int                     connectId,
        int                     reserved,
        int                     options,
        PcmsObjStruct           *objPtr,
        int                     partUid,
        int                     noRoles,
        char                    *roles[],
        char                    *userName,
        int                     *noUserRoles,
        PcmsUserRoleStruct      **ptrUserRoles
);
```

Parameters

connectId	is the database connection identifier.
reserved	is reserved for future use.
options	is reserved for future use.
objPtr	is a pointer to a change document *PcmsObjStruct* that has been initialized via *PcmsInitSpec()* or *PcmsInitUid().*
partUid	is the *partUid* for which to obtain roles.
noRoles	is the optional number of roles on which you wish to filter.
roles	is an optional array of char * roles that are used to filter the data returned.
userName	is an optional username.
noUserRoles	is the address of an integer variable to contain the number of *PcmsUserRole* structures returned.
ptrUserRoles	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsUserRole.* If no objects are found *noUserRoles* is set to zero and *ptrUserRoles* is set to (*PcmsUserRoleStruct* *) zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsGetUserRoles() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr.*

Comments

If *partUid* is zero, then the appropriate parent part of the change document is calculated and used by the function for the tree walk.

You can use the *userName* parameter to act as a filter for the user against which this function will apply. If you specify a NULL value, then all users are retrieved.

If you know which particular roles that interest you, you can use the *noRoles* and *roles* parameters to filter for these roles. If *noRoles* is zero (0), all the roles will be retrieved.

Related Functions

PcmsGetPendingUsers().

PcmsGetPendingUsers - Obtain Pending User Structures

Purpose

This function allows $CHANGE-MANAGER(S) to retrieve the list of pending users (with their roles and capabilities) for a specified change document object.

Each of the structures that are returned detail the next status and phase possible for a user on that change document.

This function supports only objects of PCMS_CHDOC type.

Prototype

```
int
PcmsGetPendingUsers (
       int                          connectId,
       int                          options,
       int                          reserved,
       PcmsObjStruct                *objPtr,
       int                          *noPendingUsers,
       PcmsPendingUserStruct        **ptrPendingUsers
);
```

Parameters

connectId	is the database connection identifier.
options	is reserved for future use.
reserved	is reserved for future use.
objPtr	is a pointer to a change document - *PcmsObjStruct* that has been initialized via *PcmsInitSpec()* or *PcmsInitUid()*.

continued

noPendingUsers	is the address of an integer variable to contain the number of *PcmsPendingUser* structures returned.
ptrPendingUsers	is a pointer to a contiguous block of allocated memory that lists the structures of type *PcmsPendingUser*. If no objects are found *noPendingUsers* is set to zero and *ptrPendingUsers* is set to (*PcmsPendingUser* *) zero. It is the responsibility of the caller application to free this pointer when it is no longer required.

Return Codes

PcmsGetPendingUsers() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Related Functions

PcmsInitSpec(), *PcmsInitUid()*, *PcmsGetUserRoles()*.

PcmsGetRSNames - Obtain Role Section Names for a Product

Purpose

This function retrieves the list of role section names corresponding to a given object uid. This uid can be for a change document, an item, a part or a type name. If you specify an object uid, then only those role sections applicable to that object are returned. If you specify a type uid, then all role sections associated with that type's lifecycle are returned.

Prototype

```
int
PcmsGetRSNames (
       int            connectId,
       int            reserved,
       int            options,
       int            uid,
       char           **message,
       int            *noValues,
       char           **values
);
```

Parameters

connectId	is the database connection identifier.
reserved	is reserved for future use.
options	is reserved for future use.
uid	is the object uid or type uid that will be used.
message	is the error message returned if the status is not PCMS_OK. It is the caller's responsibility to free this allocated memory if not NULL.

continued

noValues	is the address of an integer corresponding to the number of role section names returned.
values	is a char * array of role section names returned. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsGetRSNames() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr*.

Related Functions

PcmsGetRSAttrs().

PcmsGetRSAttrs - Retrieve Attribute Numbers in a Role Section

Purpose

This function returns the attribute numbers that are used by a given role section object/type uid combination. The order in which these attribute numbers are returned is in the display order described in the process model. If you specify an object uid, then only those roles section attributes applicable to that object are returned. If you specify a type uid, then all the role section attributes associated with that type's lifecycle are returned.

Object and type uids are mutually exclusive. If you specify object uid (*objUid*), this will cause the function to ignore any type uids that you may additionally specify.

This function currently supports items, change documents and parts.

Prototype

```
int
PcmsGetRSAttrs (
       int            connectId,
       int            reserved,
       int            options,
       int            objUid,
       int            typeUid,
       char           *roleName,
       char           *userName,
       int            *noAttrs,
       int            **attrs,
       char           **defaultRole
);
```

Parameters

connectId	is the database connection identifier.
reserved	is reserved for future use.
options	is reserved for future use.
objUid	is the uid of the object that the function will use.
typeUid	is the uid of the type name that the function will use.
roleName	is the role name to request the attribute numbers for. If a NULL string is used, the default role section name will be calculated and returned via the parameter *defaultRole*.
userName	is reserved for future use.
noAttrs	is the total number of attribute numbers returned.
attrs	is a pointer to the list of attribute numbers contained in this role section. It is the responsibility of the calling application to free this pointer when it is no longer required.
defaultRole	is the address of a char * that is used to store the default role section name when the *roleName* parameter is a NULL string. It is the caller's responsibility to free the memory if this string is not NULL.

Return Codes

PcmsGetRSAttrs() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr*.

Comments

You can use the *roleName* parameter to filter on those attributes which apply to a certain role section name. In addition, two special filters can also be used.

1	$ALL	returns all the attributes that are associated with that object or type uid's lifecycle.
2	$ALL_ROLE_SECTIONS	returns all the attributes used by any role sections for this object or type uid's lifecycle.

If you specify the *roleName* parameter as a NULL string, the function will calculate the default role section name and populate this into the *defaultRole* parameter.

Related Functions

PcmsGetRSNames()

PcmsGetUserRelTypes - Obtain User Relationship Subtypes

Purpose

This function returns all the user-relationship sub-types for a specified product. These relationship types include affected, information, dependent and user-defined item-to-item relationships.

Prototype

```
int
PcmsGetUserRelTypes (
        int                     connectId,
        int                     reserved,
        int                     options,
        char                    *productId,
        int                     *noRelTypes,
        PcmsRelTypeStruct       **relTypes
);
```

Parameters

connectId	is the database connection identifier.
reserved	is reserved for future use.
options	is reserved for future use.
productId	is the product Id that the function will use.

continued

noRelTypes	is pointer to an integer variable in which to store in *relTypes.*
relTypes	is a pointer to a contiguous block of allocated memory that lists the an array of *PcmsRelTypeStructs* returned. If no objects are found as a result of the function call, then *noRelTypes* is set to zero and *relTypes* is set to (*PcmsRelTypeStruct *)* zero. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsGetUserRelTypes() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

Comments

For completeness, definitions of the standard Dimensions-defined relationships are also given.

PcmsPopulate - Populate an Object's Attributes Values

Purpose

This function populates a given *PcmsObjStruct* with attributes, attribute definitions and values. The values are copied from an existing object which you supply, and are merged with the attributes generic to a specified type uid.

Prototype

```
int
PcmsPopulate (
       int                  connectId,
       int                  options,
       int                  objType,
       int                  typeUid,
       PcmsObjStruct        *primeObj,
       PcmsObjStruct        **outObject
 );
```

Parameters

connectId	is the database connection identifier.
options	is reserved for future use.
objType	is the type of the *outObject* e.g. PCMS_ITEM
typeUid	is the *typeUid* for the *outObject.*
primeObj	is a pointer to another object to prime the values of the *outObject* from.
outObject	is the address of a pointer to a *PcmsObjStruct* that will be allocated and populated by this function.

Return Codes

PcmsPopulate() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

PcmsGetCandidates - Retrieve Candidates for Delegation

Purpose

This function returns a list of those users who are valid candidates for the object, role and capability supplied. Currently this function supports only objects of the type PCMS_CHDOC.

Prototype

```
int
PcmsGetCandidates (
       int                  connectId,
       int                  options,
       PcmsObjStruct        *objPtr,
       char                 *role,
       char                 capability,
       char                 *noCandidates,
       char                 ***candidates
 );
```

Parameters

connectId	is the database connection identifier.
options	is reserved for future use.
objPtr	is pointer to a *PcmsObjStruct* that contains an object that has been initialized with *PcmsInitSpec()* or *PcmsInitUid()*.
role	is the role for which to retrieve candidates e.g. “DEVELOPER”.
capability	is the capability that the candidate has – 'L' (Leader),'P' (Primary) or 'S' (Secondary).

continued

noCandidates	is a pointer to an integer in which to store the number of returned values.
candidates	is the address of a pointer in which the returned list of users will be returned. The function will allocate the associated memory. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsGetCandidates() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo*, and *PcmsDbErrorStr*.

PcmsGetAttrFile - Get Change Document Descriptions

Purpose

This function enables you to obtain either the detailed description, the current action description or full action for a specified change document.

Prototype

```
int
PcmsGetAttrFile (
       int                  connectId,
       PcmsObjStruct        *objPtr,
       int                  options,
       int                  attrNo,
       char                 **toFile,
       int                  *size
 );
```

Parameters

connectId is the database connection identifier.

objPtr is the pointer to the change document that the function will use.

options Options 0 to 2 are used as follows:

- 0 Copies the selected description into a file specified by the variable *toFile*
- 1 Copies the selected description into the variable *toFile* and populates the size of the description file into the variable *size.*

continued

NOTE that if *toFile* is a NULL pointer, then this function will allocate memory to hold the description. It is then the caller's responsibility to free this memory.

2 Places only the size of the description into the variable *size.*

attrNo specifies the type of description requested, the following values are valid.

PCMS_ATTR_ACTION_DESC	Returns the full action description of the change document
PCMS_ATTR_THIS_ACTION_DESC	Returns the current action description for the change document
PCMS_ATTR_DETAIL_DESC	Returns the detailed description of the change document

toFile is a pointer to where the description is to be copied. If **toFile* is a NULL pointer, this function will allocate memory to hold the description. It is the responsibility of the caller to free this memory after use.

size is a pointer to an integer into which the description size is returned.

Return Codes

PcmsGetAttrFile() returns:

PCMS_OK	on success
PCMS_FAIL	on failure to find any data
PCMS_ERROR	on failure and sets *PcmsErrorNo*, *PcmsErrorStr*, *PcmsDbErrorNo* and *PcmsDbErrorStr*.

PcmsEvntFree – Free Memory

Purpose

This function is a wrapper to the C function *free()*. It must be used to free memory allocated by DTK functions, such as *PcmsQuery().* The reason for this is that on some platforms, like Windows NT/ 2000, the memory that is allocated within a shared library **must** be freed by the same shared library. If this is not done, then memory errors begin to occur.

Prototype

```
void PcmsEvntFree (void *ptr);
```

Parameters

ptr is a pointer to the memory block that will be freed.

PcmsEvntMalloc – Allocate Memory

Purpose

This function is a wrapper to the C function *malloc*. It must be used to allocate memory within a DTK application. The reason for this is that on some platforms, like Windows NT/2000, the memory that is allocated within a shared library **must** be freed by the same shared library. If this is not done, then memory errors begin to occur. This function is thus provided as a convenience to ensure that all the memory is allocated within the context of the same shared library.

Prototype

```
void* PcmsEvntMalloc (int size);
```

Parameters

size is the size of the memory to allocate.

PcmsEvntCalloc – Allocate Zero Initialized Memory

Purpose

This function is a wrapper to the C function *calloc()*. It must be used to allocated zero initialized memory within a DTK application. The reason for this is that on some platforms, like Windows NT/2000, the memory that is allocated within a shared library **must** be freed in the same shared library. If this is not done, then memory errors begin to occur. This function is thus provided as a convenience to ensure that all the memory is allocated within the context of the same shared library.

Prototype

```
void* PcmsEvntCalloc (int size);
```

Parameters

size is the size of the memory to allocate.

PcmsEvntRealloc – Re-allocate Memory

Purpose

This function is a wrapper to the C function *realloc()*. It must be used to re-allocate memory within a DTK application. The reason for this is that on some platforms, like Windows NT/2000, the memory that is allocated within a shared library **must** be re-allocated by the same shared library. If this is not done, then memory errors begin to occur. This function is thus provided as a convenience to ensure that all the memory is maintained within the context of the same shared library.

Prototype

```
void* PcmsEvntRealloc (void *ptr, int size);
```

Parameters

ptr	is a pointer to the block of memory that will be resized.
size	is the size of the new memory.

PcmsGetCommandLine – Get the Dimensions Command

Purpose

This function will return a constant string pointer to a copy of the command that was submitted to Dimensions. This is intended to allow you, from within a DTK event, to determine what Dimensions command was actually run.

Prototype

```
const char* PcmsGetCommandLine (void);
```

Parameters

None

Attribute Macros

Defined in the file *pcms_api.h* are a set of macros which have been provided to help you in writing your application. These macros are public but the structures that they use may change in the future and should not be used directly.

Initialize *PcmsObjStruct* attrs

- **PcmsInitAttrStruct(objPtr, number)**

  This macro allocates number zero initialized memory structure of type PcmsObjAttrStruct to the attrs pointer and updates noAttrs accordingly.

  The parameters are:

objPtr	a pointer to a *PcmsObjStruct.*
number	the number of *PcmsObjAttrStruct*(s) for which to allocate memory.

  NOTE This macro works only on *PcmsObjStruct*s that have not had attributes already setup.

Add *attrDef* Structures

- ***PcmsAddAttrDefs(objPtr)***

  This macro allocates zero memory for all the *PcmsObjAttrDefStruct*(s) within a *PcmsObjStruct.*

  The parameter is:

objPtr	a pointer to a *PcmsObjStruc*.

  NOTE This macro works only on *PcmsObjStructs* that have NULL *objPtr.attr[n].attrDef*s.

Single-Value Attributes (SVA)

- ***PcmsSvaSetVal (valuePtr, string, reserved)***

  This macro sets an attribute value in a single-value attribute.

  The parameters are:

valuePtr	the value structure being initialized and set (for a *PcmsObjStruct* object *objPtr* and attribute number n, *valuePtr* is *objPtr.attrs[n].value*).
string	the value itself expressed as a char * string.
reserved	is an integer field reserved for future use (use 0).

- ***PcmsSvaReSetVal (valuePtr, string, reserved)***

  This macro resets an attribute value in a single-value attribute.

  The parameters are:

valuePtr	the value structure being initialized and set (for a *PcmsObjStruct* object *objPtr* and attribute number n, *valuePtr* is *objPtr.attrs[n].value*).
string	the value itself expressed as a char * string.
Reserved	is an integer field reserved for future use (use 0).

- ***PcmsSvaGetVal (valuePtr)***

  This macro returns a char * corresponding to the string value of this attribute.

  The parameter is:

valuePtr	the value structure pointer being queried (for an object *objPtr* and attribute number n, *valuePtr* is *objPtr.attrs[n].value*).

Multi-Value Attributes (MVA)

Instead of a single value, multiple-valued attributes maintain a value-set which is accessed through the use of an index. You can use the following macros to access and set this value-set.

- ***PcmsMvaSetVal (valueSetPtr, index, string, reserved)***

  This macro appends a value to a value-set.

  The parameters are:

valueSetPtr	the value set being added to (for an object *objPtr* and attribute number n the *valueSetPtr* is *objPtr.attrs[n].value*).
index	the index into the list. This index is incremented within the macro. For the first value in the value-set, this index is zero.
string	the value itself as a char * string.
reserved	is an integer field reserved for future use (use 0).

- ***PcmsMvaNumVals(valueSetPtr)***

  This macro returns an integer value corresponding to the number of values currently in the value-set.

  The parameter is:

valueSetPtr	the value set to query.

- ***PcmsMvaGetVal (valueSetPtr, index)***

  This macro returns the value of the value-set at a specific *index* as a char * string.

  The parameters are:

valueSetPtr	the *valueSet* to query.
index	the integer index in the list of this attribute's value-set.

- ***PcmsMvaReSetVal(valueSetPtr, index, string,reserved)***

  This macro frees a certain indexed value and writes the new string into the same position given by the index.

  The parameters are:

valueSetPtr	the value-set to manipulate.
index	the integer index in the list of this attribute's value-set value.
string	the new string.
reserved	reserved

- ***PcmsMvaFree (valueSetPtr)***

  This macro frees a complete *PcmsMva* i.e. frees the whole list.

  The parameters is:

valueSetPtr	the *valueSet* to free.

NOTE It is useful to note that the SVA macros are only a convenience. It is possible to access all attribute value structures with *PcmsMvaNumVals()* and *PcmsMvaGetVal()*. For single value attributes *PcmsMvaNumVals()* will return 1.

---

4 DTK API Functions for Win32 Client Installations

In this Chapter

Introduction

This chapter describes a set of functions that have been specifically written for Windows 98/NT/2000 client machines that have had only the Dimensions Windows clients (CD-2) installed. These functions give you access to the full functionality of the DTK but are specifically written for Win32 clients.

The description of each function has the following components

Purpose	What the function does
Prototype	The function prototype
Parameters	Description of the parameters used in the function
Return Codes	Codes returned (please refer to DTK return codes on page 20 for further details)
Sample	Sample function call (if applicable)
Comments	Additional relevant information (if applicable)
Related Functions	Any related DTK function calls

Building Client Applications

These functions are available in the supplied *clientapi.h* and *clientapi.lib* files located in the directory *"<Dimensions_Root>\pcms_api\"*. Any source file that references the functions or constants must include this file.

If your application references connection functions, such as *PcmsClntApiConnect()*, you must include the standard WinD2K file `windows.h` to properly compile the application.

NOTE Starting with Dimensions 7.1, the `pcms_api.lib` library file has been renamed to `pcms_api`*XX*`.lib`, where *XX* is the version number of the Dimensions release. For example, for Dimensions 7.1, the file is named `pcms_api71.lib`.

Sample Code Fragment

```
PcmsObjStruct pObj = { 0 };
int conId = 0;
char errBuff[1024];

/* API will now connect and show the login dialog */
conId = PcmsClntConnect((HWND)NULL);
/* This is an example public API call looking for the Workset
   "TEST_WORKSET" */
if (PcmsInitSpec(conId, "TEST:TEST_WORKSET",
    PCMS_WORKSET, &pObj)!=PCMS_OK)
{
    /* API called failed so get the error and display it */
    (void)PcmsClntGetLastError(conId, errBuff,
                sizeof(errBuff));
    (void)fprintf(stderr, errBuff);
    return;
}
/* List the directories in the Workset "TEST_WORKSET" */
(void)PcmsClntExecCommand(conId, "LWSD TEST:TEST_WORKSET");
/* Now display the output */
(void)PcmsClntGetLastError(conId, errBuff, sizeof(errBuff));
(void)fprintf(stdout, errBuff);
/* Now disconnect */
(void)PcmsClntApiDisconnect(conId);
```

PcmsClntApiConnect - Connect to a Dimensions Database

Purpose

This function provides you with a Login window allowing you to connect to a Dimensions database. This function will return a *connectId* (integer) that represents your database connection.

Prototype

```
int
PcmsClntApiConnect
(
        HWND parent=NULL
);
```

Parameters

HWND parent	is the parent window for the login dialog.

Return Codes

PcmsClntApiConnect() returns:

connectId	on successfully completing the connection to the Dimensions server.
PCMS_ERROR	on error
PCMS_FAIL	on failure

Related Functions

PcmsClntApiSilentConnect()

PcmsClntApiSilentConnect - Connect Silently to a Dimensions Database

Purpose

This function provides you with a connection to a Dimensions database as specified by your input parameters. This function will return a *conId* that represents your connection to the Dimensions Server.

Prototype

```
int
PcmsClntApiSilentConnect
(
        char    *user,
        char    *password,
        char    *host,
        char    *pcms_install,
        char    *db_name,
        char    *db_pword,
        char    *db_node
);
```

Parameters

user	is your operating system login name.
password	is your operating system password.
host	is the Dimensions server node name that you wish to connect to.
pcms_install	is the Dimensions server installation directory e.g. `/usr/pcms/dimensions7_1/`.

continued

db_name	is the name of the Dimensions database that you wish to connect to e.g. *pcms_tool*.
db_pword	is the password of the database.
db_node	is the ORACLE service name assigned to the node where the oracle database is located.

Return Codes

PcmsClntApiSilentConnect() returns:

int connectId	on successful connection
PCMS_ERROR	on error
PCMS_FAIL	on failure

Related Functions

PcmsClntApiConnect()

PcmsClntApiDisconnect - Disconnect from a Dimensions Database

Purpose

This function disconnects from the Dimensions database as specified by the conId. The *connectId* must be a valid *connectId* returned by *PcmsClntApiConnect()* or *PcmsClntApiSilentConnect().*

Prototype

```
int
PcmsClntApiDisconnect
(
        int     connectId
);
```

Parameters

connectId	is the database connection identifier.

Return Codes

PcmsClntApiDisconnect() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Related Functions

PcmsClntApiConnect(), PcmsClntApiSilentConnect()

PcmsClntApiGetLastError - Get the Last Dimensions Message

Purpose

This function allows you to access the output from the last Dimensions command that was run on the server via *PcmsClntApiExecCommand()*.

Prototype

```
int
PcmsClntApiGetLastError
(
       int     connectId,
       char    *errorBuffer,
       int     maxLength
);
```

Parameters

connectId	is the database connection identifier.
errorBuffer	is a pointer to a user allocated character array that is populated with the text of the server message.
maxLength	is the maximum length of the server message to be displayed.

Return Codes

PcmsClntApiGetLastError() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Related Function

PcmsClntApiGetLastErrorEx()

PcmsClntApiGetLastErrorEx - Get the Last Dimensions Message

Purpose

This function allows you to access the output from the last Dimensions command that was run on the server via *PcmsClntApiExecCommand()*. The functionality of this command is the same as for *PcmsClntApiGetLastError()* except that the buffer size is dynamically allocated.

Prototype

```
int
PcmsClntApiGetLastErrorEx
(
        int     connectId,
        char    **errorBuffer
);
```

Parameters

connectId	is the database connection identifier.
errorBuffer	is a pointer to a contiguous block of allocated memory that is populated with the message text. It is the responsibility of the calling application to free this pointer when it is no longer required.

Return Codes

PcmsClntApiGetLastErrorEx() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Related Function

PcmsClntApiGetLastError()

PcmsClntApiModeBinary - Set File Transfer Mode to Binary

Purpose

This function sets the file transfer format to binary for any subsequent item commands that are issued.

Prototype

```
int
PcmsClntApiModeBinary
(
        int     connectId
);
```

Parameters

connectId is the database connection identifier.

Return Codes

PcmsClntApiModeBinary() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Comments

You must use this command if you are going to perform operations that require file transfer from the client to the server (or visa versa) to occur in binary mode. An example of this might be getting a binary item into your PC. When you perform any file transfer operations, such as check in (RI), Dimensions will not validate the format of the file being transferred.

PcmsClntApiModeText - Set File Transfer Mode to ASCII

Purpose

This function sets the file transfer format to ASCII for any subsequent item commands that are issued.

Prototype

```
int
PcmsClntApiModeBinary
(
        int     connectId
);
```

Parameters

connectId is the database connection identifier.

Return Codes

PcmsClntApiModeText() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Comments

You must use this command if you are going to perform operations that require file transfer from the client to the server (or visa versa) to occur in ASCII mode. An example of this might be getting an ASCII item into your PC. When you perform any file transfer operations, such as check in (RI), Dimensions will not validate the format of the file being transferred.

PcmsClntApiFree – Free Memory

Purpose

This function is a wrapper to the C function *free()*. It must be used to free memory allocated by PcmsClnt API DTK functions, such as *PcmsClntApiGetLastErrorEx().* The reason for this is that on some platforms, like Windows NT/2000, the memory that is allocated within a shared library MUST be freed by the same shared library. If this is not done, then memory errors begin to occur.

Prototype

```
void PcmsClntApiFree (void *buffer);
```

Parameters

buffer is a pointer to the memory block that will be freed.

PcmsClntApiExecCommand - Execute a Dimensions Command

Purpose

This function sends a command to the Dimensions server specified by *connectId.*

Prototype

```
int
PcmsClntApiExecCommand
(
        int     conId,
        char    *command
);
```

Parameters

connectId	is the database connection identifier.
command	is a pointer to a user allocated character array that is populated with the command to be executed.

Return Codes

PcmsClntApiExecCommand() returns:

PCMS_OK	on success
PCMS_ERROR	on error
PCMS_FAIL	on failure

Additional Supported DTK Functions

In addition to the specific Win32 functions described in this chapter, the following standard Dimensions DTK functions are also available. These functions are fully documented in Chapter 3.

*PcmsObjGetRels() **	*PcmsLovFree()*
*PcmsObjGetBackRels() **	*PcmsGetRSNamest() **
*PcmsCntrlPlanGet() **	*PcmsGetRSAttrs() **
*PcmsQuery() **	*PcmsGetUserRelTypes() **
*PcmsPendGet() **	*PcmsFullQuery() **
PcmsInitSpec()	*PcmsGetUserRoles() **
PcmsInitUid()	*PcmsGetCandidates() **
PcmsGetAttrs()	*PcmsGetPendingUsers() **
PcmsObjFree()	*PcmsGetWsetObj() **
PcmsPendWhoGet()	*PcmsEvntMalloc()*
PcmsAttrDefInit()	*PcmsEvntCalloc()*
*PcmsGetAttrDefNum() **	*PcmsEvntRealloc()*
*PcmsPopulate() **	*PcmsAttrGetLov() **
PcmsEvntFree()	*PcmsAttrValidate() **

* **NOTE** Functions marked with an asterisk (*) require that the following file be installed on each client: *Dimensions_Root*\msg\pcms_api_sql_uk.msb. This can be accomplished by a default client installation or by copying the file from the Dimensions server, where it resides in the same directory.

5 Dimensions Events Callout Interface

In this Chapter

Description

This chapter outlines the functionality, design and implementation of applications using the Dimensions Event Callout Interface. Before reading this chapter ensure that you familiarize yourself with the concept of shared libraries because it is via this mechanism that this event interface is implemented.

Shared Libraries

The Dimensions Event Callout Interface is provided by giving you access to a public function call that is invoked when certain Dimensions commands are run. This function is called *userSuppliedFunction()* and is resolved in a shared library called *libpcmsu.* On a default Dimensions installation a stub version of this library is provided. If you wish to implement your own Event Callout, you will need to build your own shared library and use this in place of the stub.

For more information on how to build shared libraries please refer to your system documentation or for guidelines refer to the examples provided in:

- "*<Dimensions_ROOT>/pcms_api/examples/*" for UNIX
- "*<Dimensions_ROOT>**pcms_api**examples*\" for Windows.

Public Function Call

The prototype for the public function call *userSuppliedFunction()* is:

```
int     userSuppliedFunction(
                PcmsEventStruct        *ptrPcmsEventStruct,
                PcmsObjStruct          *ptrObj,
                PcmsObjStruct          *ptrUser,
                char                   **ptrErrorMessage,
                int                    *noEventInfo,
                void                   **ptrEventInfo);
```

where the parameters are:

ptrPcmsEventStruct	is a pointer to a *PcmsEventStruct* in which the details on the current event being fired are held.
ptrObj	is a pointer to a *PcmsObjStruct* which holds the object details pertaining to the Dimensions' object currently being processed.
ptrUser	is a pointer to a *PcmsObjStruct* which holds the details on the user currently running the event.
ptrErrorMessage	is a pointer to a pointer which allows you to setup an error message to be printed instead of the default Dimensions message.

continued

noEventInfo	is an integer variable which is used in context with *ptrEventInfo* to access members of that pointer.
ptrEventInfo	is a pointer to a void *pointer which is populated with different information depending on the event called. For PCMS_EVENT_MAIL, PCMS_EVENT_DLGI and PCMS_EVENT_DLGC events this pointer will point to an array of *PcmsUserRoleStruct*s. For a PCMS_EVENT_RELATE or PCMS_EVENT_UNRELATE event this pointer will point to an array of *PcmsRelStruct*s.

Event Callout Interface

As indicated previously, when certain Dimensions commands are run the public function *userSuppliedFunction()* is invoked with a number of parameters. These parameters are used to specify what event is being fired; what Dimensions object is being affected, and finally which type of Dimensions command is being run. Each time an event is fired the following hierarchy of calls is made to *userSuppliedFunction()*:

- Validate event call (fired ONLY when a user or system attribute has changed)
- Pre-event call
- Post-event call.

Each of the event calls allows you to perform a number of operations on the Dimensions object on which the event has been called.

You can access the type of event being fired by examining the *ptrPcmsEventStruct* as described on page 168.

Validate Events

Validate events are called prior to any Dimensions validation being run on the data supplied. Typically, you can use validation events to inspect information (such as the object details, default and user attributes) and then change this information. You could, for example, use this event to implement your own automatic item id generator or perform extra validation on user-defined date attributes. Once this event has been fired Dimensions will then proceed with its normal validation checks. This event is indicated by the use of the constant PCMS_EVENT_VALIDATE_OP.

NOTE Validate events are fired ONLY when user or system attributes change as a result of a command.

Pre-events

Pre-events are called prior to the Dimensions command being executed. You can use this event to stop the execution of this command by returning the failure status PCMS_FAIL. If you populate the *ptrErrorMessage*, then this error string will be displayed via whatever interface invoked the command. Typically, you can use this event to perform any specific validation before deciding to let the command proceed. This event is indicated by the use of the constant PCMS_EVENT_PRE_OP.

Post-events

Post-events are called after the Dimensions command has been run. Typically you can use this event to perform any 'clean up' or post-command logging to other applications. This event is indicated by the use of the constant PCMS_EVENT_POST_OP.

Event Types

In addition to the event hierarchy described in the previous sub-sections, each event fired also has an event type which relates to the type of Dimensions command being run. These event types are:

Event Type	Activity to a Dimensions Object
PCMS_EVENT_ACTION	Actioned
PCMS_EVENT_CREATE	Created
PCMS_EVENT_CANCEL	Check out of object is canceled
PCMS_EVENT_DELETE	Deleted
PCMS_EVENT_DLGC	Change document is delegated to a user
PCMS_EVENT_DLGI	Item is delegated to a user
PCMS_EVENT_EXTRACT	Object is checked out (or extracted)
PCMS_EVENT_FETCH	Object is gotten (fetched) (or browsed)
PCMS_EVENT_MAIL	A Dimensions mail message is being sent
PCMS_EVENT_RELATE	Object is related to another object
PCMS_EVENT_UNRELATE	Object is unrelated from another object
PCMS_EVENT_RETURN	Object is checked in (or returned)
PCMS_EVENT_UPDATE	Updated

While a single Dimensions command, such as ‘getting (fetching) an item’, can fire a single event type (PCMS_EVENT_FETCH) it is also possible for a command to generate multiple different events. Consider, for example, the following command:

```
EDI  “FS:TEST.A-SRC,1” /REV=2.1/ATTRIBUTE=(COMPLEXITY=“Delta7”)
```

This command performs a check out, check in, and an update of attribute values. Thus, if you ran the command as described previously, the following events would be fired.

Event Type	Action to a Dimensions Item
PCMS_EVENT_EXTRACT	Checked out from Dimensions
PCMS_EVENT_UPDATE	Attributes updated
PCMS_EVENT_RETURN	Checked in to Dimensions
PCMS_EVENT_MAIL	Notification of new item sent via e-mail

If you added additional parameters to this command, such as /STATUS or /RELATE_CHDOC, other events would also have been fired.

The table below describes the commands which fire events and the objects types they involve.

EventId	PART	ITEM	CHDOC	WKSET	BASELINE	RELEASE
PCMS_EVENT_						
— CREATE	CP, CPV	CI, IP	CC	DWS, MWS	CBL, CMB, CRB	REL
— EXTRACT	UP	EI, EDI, UI, IP				
— RETURN	UP	RI, EDI, UI, IP				
— UPDATE	UPA, UP, CP, CPV	CI, EI, EDI, RI, UI, UIA, IP	CC, UC			
— DELETE	DPV	DI, PUR		RWS	DLB	DREL
— ACTION	SPV, UP	AI, SI	AC			

EventId	PART	ITEM	CHDOC	WKSET	BASELINE	RELEASE
— FETCH		FI	BC			
— RELATE	RPCD	RICD	RCCD			
— UNRELATE	XPCD	XICD	XCCD			
— MAIL	(N/A)	AI, CI, RI, UI	AC			
— DLGC	(N/A)		DLGC			
— CANCEL	(N/A)	CIU				
— DLGI		DLGI				

Determining the Event you want

When the *userSuppliedFunction()* is invoked, one of the parameters passed in is a pointer to a *PcmsEventStruct* that can be interrogated for details such as:

- the event type e.g. PCMS_EVENT_CREATE
- where in the hierarchy the event is being fired e.g. validate or pre-event
- the object type the event is being fired on e.g. PCMS_ITEM.

By examining the following fields of the *PcmsEventStruct* you can trap the event you specifically require.

eventId	the event type
whenCalled	the position in the event hierarchy
objType	the type of object that the event was fired on.

First and Second Event Calls

Populating all the parameters for an event can take time, especially if you are connected to a remote database over a WAN. As a result of this, each time an event is to be fired you get a chance to determine whether or not you are really interested in that event. This separation is known as the 'first and second call' to an event.

The first call to an event populates only the sparse details on the Dimensions object and the *PcmsEventStruct* to allow you to determine if you are interested in this event. The specific details provided in the PcmsObjStruct depend on what command is called, on what object, and under what circumstances--as shown in the next two tables.

Table 5-1. First call details for object creation operations

Object type	First call details
Change Docs	objType, productId, typeName, status, typeUid
Items	objType, typeUid, typeName, productId, objId, revision, userName, status, dateTime
Baselines	objType, typeUid, typeName, productId, objId, variant, revision, userName, status, dateTime
Parts	objType, typeUid, typeName, productId, objId, userName, dateTime
Releases	objType, productId, objId
Worksets	objType, typeUid, typeName, productId, objId, variant, revision, description, userName, status, dateTime

Table 5-2. First call details for other operations

Object type	First call details
Change Docs	uid, objType, typeUid, typeName, productId, objId, userName, status, dateTime
Items	uid, objType, typeUid, typeName, productId, objId, variant, revision, description, userName, status, dateTime
Baselines	uid, objType, typeUid, typeName, productId, objId, variant, revision, userName, status, dateTime
Parts	uid, objType, typeUid, typeName, productId, objId, variant, revision, description, userName, status, dateTime
Releases	uid, objType, productId, objId, dateTime
Worksets	uid, objType, typeUid, typeName, productId, objId, variant, revision, description, userName, dateTime

A first call event can be determined by checking if:

```
ptrUser == (PcmsObjStruct *)0 AND ptrErrorMessage == (char *)0
```

If you select the first call, then only the *ptrPcmsEventStruct* will be fully populated. The *ptrObj* will only have the object-specification fields and uid filled in. If as a result of examining these partial details, you decide that you really want this event, then returning the status PCMS_OK will generate a second event call. If you are not interested in this event, then return PCMS_FAIL and the second call to this event will not be fired.

The second event call can be regarded as the 'real' event call. This has all the data structures filled in and allows you to access all the attribute information for the object. It is on this call that your event should perform its operation.

Event Call Summary

The following table summarizes the event callout mechanism.

Step	Command
1	**Read Dimensions Command**
2	**FIRST CALL VALIDATE EVENT and check that VALIDATE EVENT is required**
3	**If VALIDATE EVENT is required, then call VALIDATE EVENT again**
4	**If VALIDATE EVENT not required, then END**
5	**Evaluate User Data and Execute the Dimensions Command**
6	**FIRST CALL PRE-EVENT and check that PRE-EVENT is required**
7	**If PRE-EVENT is required, then call PRE-EVENT again with fully detailed data**
8	**If PRE-EVENT not required, then END**
9	**COMMIT Dimensions Command to database**
10	**FIRST CALL POST-EVENT and check that POST-EVENT is required**
11	**If POST-EVENT is required, call POST-EVENT again with fully detailed data**

Steps 3, 7 and 11 occur when the real VALIDATE, PRE and POST EVENTS are done.

Writing a DTK Callout Event

This section describes how to design and write a DTK event, the pitfalls to watch out for, and in what situations an event is applicable.

Is an Event the Solution for you?

Before you start to write an event to implement your solution, you must decide if an event is what you really need. To help you decide bear in mind the following questions.

- Is the functionality that you seek to achieve already in Dimensions?

  If you are seeking to implement stronger rules for object relationships or attributes, this functionality is already available in Dimensions.

- Is the functionality that you seek to achieve initiated by a Dimensions command?

  The events are strictly intended to allow you to perform inline processing or checking on a specific object. They are not intended to allow you to run multiple Dimensions commands on the same object. When a validate-event or pre-event is fired on an object, then that object becomes locked until the transaction has been committed to the database. If you try to spawn another Dimensions command, then you run the following risks.

  - The Dimensions command that you spawned will be suspended waiting for the lock to be released; and it never will be released until a time-out has occurred.
  - The Dimensions command will cause the same event to be fired that will spawn yet another Dimensions command that calls the same event spawning yet another command, etc., and so on until your machine locks up.

  If you are intending to use an event to spawn other Dimensions-related commands, then you are strongly advised to use a separate DTK client application to perform this sequencing of commands.

- Does the operation you want to capture actually fire an event?

Not all Dimensions commands fire events. You need to be sure that the operation you want to capture actually fires an event.

Designing your Event

When you have decided that your solution requires an event, you need to decide which event you need to capture, and which type of event you have to use. Events are fired when you run a Dimensions command, so decide on the list of Dimensions commands that invoke the events you want and scope this list to the type of objects you wish to process. Examining this list you may discover that you may need to filter out certain events, commands or objects to obtain just the processing you want. You will need to code this filtering into your event. When you are looking at this list remember that additional parameters can call additional events. You can filter the events that you select by either examining the *ptrPcmsEventStruct* as described previously in "“Public Function Call” on page 163, or you can access the actual Dimensions command being run via *PcmsGetCommandLine()* and filter on this.

Once you know the list of commands and type of events that you are going to trap, you need to consider where in the callout hierarchy this trapping will occur. The basic rules are to trap:

- the validate event if you wish to change any information
- the pre-event if you wish to be able to stop the operation
- the post-event if you wish to perform an action after the Dimensions operation has committed data to the database and freed all the locks on that object.

Suggestions for More Common Operations

- Changing attributes (user and system) or setting defaults

  To change, set or reformat attributes you have to capture the PCMS_EVENT_UPDATE at the PCMS_EVENT_VALIDATE_OP stage in the call hierarchy.

- Changing object Id on creation

  If you wish to wish to change the object identifier or filename used when an object is created, then you have to capture the PCMS_EVENT_CREATE and PCMS_EVENT_UPDATE events at the PCMS_EVENT_VALIDATE_OP stage in the call hierarchy.

- Checking user files before they are returned (checked in) to Dimensions

  If you wanted to perform some custom formatting or checks on user files before they are returned to Dimensions, then you would need to capture the PCMS_EVENT_RETURN event at the PCMS_EVENT_PRE_OP stage in the call hierarchy.

- Allowing the action of an object only if certain criteria are matched

  If you have specific checks that you wish to perform before objects are actioned from one state to another e.g. releasing a baseline to test, then you have to capture the PCMS_EVENT_ACTION event at the PCMS_EVENT_PRE_OP stage in the call hierarchy.

- Logging to another application that a Dimensions operation has occurred

  If you have integrated Dimensions with another application and wish to signal to that application that a command has been run, then you have to capture all the events at the PCMS_EVENT_POST_OP stage in the call hierarchy.

Writing your Event

You have to write your event code with the *userSuppliedFunction()* as the interface point between Dimensions and your event code. The return codes from this function call are the standard PCMS_OK, PCMS_FAIL and PCMS_ERROR. These return calls have the following effects:

First Event Call	
PCMS_OK	causes a second event call to occur.
PCMS_FAIL	causes Dimensions to ignore this event and continue with its normal processing for the operation; no second event call will be made.

Second Event Call	
PCMS_EVENT_VALIDATE_OP	PCMS_OK will allow the Dimensions operation to continue. If any attributes have been changed, then the new values will be used. PCMS_FAIL will cause the Dimensions operation to fail and any error messages in the *ptrPcmsErrorMessage* pointer will be printed. PCMS_ERROR is the same as PCMS_FAIL
PCMS_EVENT_PRE_OP	PCMS_OK will allow the Dimensions operation to continue.

continued

Second Event Call	
	PCMS_FAIL will cause the Dimensions operation to fail and any error messages in the *ptrPcmsErrorMessage* pointer will be printed. PCMS_ERROR is the same as PCMS_FAIL.
PCMS_EVENT_POST_OP	Because the operation has been completed, the status at this point is irrelevant, but for consistency you should return PCMS_OK.

In the first event call you need to interrogate the *ptrPcmsEventStruct* to determine whether or not to trap this event and, if so, return PCMS_OK, else return PCMS_FAIL.

In the second event call you need to place the code to support your event logic and return the appropriate status.

DTK Event Internals

When an event is passed to your function you are able to both manipulate the data supplied and/or view many of the internal changes that have occurred or will occur on an object as a result of the Dimensions command. This section discusses what information these pointers give you and how you can use them to achieve various different effects.

- The *PcmsEventStruct* pointer – *ptrPcmsEventStruct*

  This pointer is the most important structure passed down to an event. It is used to both control the event operation and also to indicate what attributes (both user and system) have been affected. How to determine which event is being called

has already been discussed. How the attributes are controlled is determined via the *noAttrsChanged* and *attrsChanged* members of this pointer.

When events are fired, any system or user-defined attributes that have been modified as a result of the command are populated into the *noAttrsChanged* and *attrsChanged* members. On a validate-event you can manipulate these variables to add, reset or remove attribute values.

- If you are resetting an attribute value, then loop through the *attrsChanged* pointer until you find the attribute structure that you wish to change. Once you have found this attribute, you can use the MVA or SVA macros to reset the attribute value. If you are resetting values on a MVA attribute, then you must first free the memory associated with this attribute via *PcmsMvaFree()* before adding your new values.
- If you are adding a new attribute value, then you will need to:
  - resize the *attrsChanged* pointer to add a new *PcmsObjAttrStruct,*
  - increment the *noAttrsChanged* index by 1
  - set the appropriate values on the new attribute structure.

  It is important to note that the *attrDef* pointer in the attribute structure must be set to NULL.
- If you are removing an attribute definition, then you will need to resize the *attrsChanged* pointer to remove this attribute and decrease the *noAttrsChanged* index by 1.

■ The *PcmsObjStruct* – pointer *ptrObj*

This pointer contains all the details on the object that the event is currently processing. On a first call to the event this

object contains only minimal data. On the second call to the event this object is fully populated.

When you create a new object, such as an item or a part, a validate-event is fired during which you can manipulate the contents of this pointer to change the object's details. You are able to change entries in the *objId, variant* and *revision* fields. Using this mechanism you could write an event that changes item Ids to suit your own automatic object identity generator.

- The *PcmsObjStruct* – pointer *ptrUser*

  This pointer contains all the details on the user currently running the event. This pointer is populated only on the second call and is 'read only'.

- The error pointer – *ptrErrorMessage*

  This pointer allows you to setup an error message that will be printed by Dimensions when a validate-event or pre-event returns a status other than PCMS_OK. This allows you to print your own custom error messages when an event fails.

- The *noEventInfo* and *ptrEventInfo* pointers

  These pointers operate together. Their usage is described on page 182.

Changing System Attributes on Validate Events

While there are no restrictions on changing user-defined attributes in the validate event, you are, however, limited to what system attributes you can change. While you can modify the *attrsChanged* pointer to include any system attribute definition, only the following will have any affect.

Event Type PCMS_EVENT_ :	Object Type PCMS_ :	System Attribute PCMS_ATTR_ :
CREATE	ITEM	FORMAT
		FILENAME
		LIB_FILENAME
		DIRPATH
		USER_FILENAME
	PART	PARTNO
		LOCALNO
	RELEASE	DIRPATH
EXTRACT	ITEM	FORMAT
		USER_FILENAME
RETURN	ITEM	FORMAT
		USER_FILENAME

If you specify any other values for system attributes, these will be ignored.

The above-mentioned attributes can be changed only when a new object is created or an item is checked out at a new revision.

Changing User Attributes on Validate Events

There are no restrictions on what you can do with user-defined attributes on a validate event. However, Dimensions will apply the usual validation to any attributes that you setup in the *attrsChanged* structure. If the attribute is not defined, has the wrong value or the user does not have the role to change it, then Dimensions will generate an appropriate error message.

Recommendations on how to Change Attribute Values

The following steps provide a recommended approach on how you should change the *attrsChanged* pointer.

- Examine the *attrsChanged* structure in the *ptrPcmsEventStruct.*
- If the pointer is NULL, you need to allocate memory to this pointer (i.e. the size of *PcmsObjAttrStruct)* and set *noAttrsChanged* to 1. Once the memory has been allocated, then set the members of the *attrsChanged* pointer to the appropriate values.

  For example, for an SVA

  ```
  PtrPcmsEventStruct->attrsChanged[0].attr=<ATTR_NO>
  PcmsSvaSetValue(ptrPcmsEventStruct->attrChanged[0].value,
                  ”text String”,0);
  ptrPcmsEventStruct->attrChanged[0].attrDef =
                  (PcmsObjAttrDefStruct *)0;
  ```

- If the pointer currently has attributes setup, then you need to check if the attribute you want to change is currently in that pointer. You can do this by looping through the *attrChanged[x].attr*(s) and looking for a match to your attribute number. Once you have found a match then use *PcmsSvaSetVal()*, or *PcmsMvaReSetVal()*to reset the value. If you are resetting the MVA values for an attribute, then

remember to free the attribute value set first via *PcmsMvaFree()*.

- If the pointer currently has attributes setup, but you cannot find a match using the search method indicated above then you need to re-allocate memory to the *attrsChanged* pointer to add a new *PcmsObjAttrStruct*. Using this newly allocated structure you can set the attribute values as described above and increment the *noAttrsChanged* variable by 1.

Calling DTK Functions within Events

When you call DTK functions from within an event the connection identifier (*conId*) that you need to use is 0. This is a special connection identifier that relates to the current connection that Dimensions has to your database. You do not need to call *PcmsConnect()* or *PcmsDisconnect()* to access the DTK functions. If you try to use these functions to initiate a connection to the database or another database, then your Dimensions session may become unstable.

Specialist DTK Event Functions

There are a number of DTK functions that, although available to DTK client applications, are specifically aimed at helping you to write your event. These functions are aimed at memory management and accessing the Dimensions command line.

DTK Function	Description
PcmsEvntFree()	Wrapper to *free()*
PcmsEvntMalloc()	Wrapper to *malloc()*
PcmsEvntCalloc()	Wrapper to *calloc()*
PcmsEvntRealloc()	Wrapper to *realloc()*
PcmsGetCommandLine()	Access to the Dimension's command that invoked this event.

Unsupported DTK Function Calls from within an Event

You can call virtually all the DTK functions from within an event. There are, however, a number of exceptions to this rule. Some of the DTK functions, due to the nature of the command that they are running, are not allowed to be called from within an event. These functions are listed below.

DTK Function	Description
PcmsConnect()	Connect to a Dimensions database
PcmsDisconnect()	Disconnect from a Dimensions database
PcmsExecCommand()	Execute a Dimensions command
PcmsSendCommand()	Execute a Dimensions command
PcmsSetDirectory()	Change working directory
PcmsSetDbErrorCallback()	Set callback functions
PcmsSetCallback()	Set callback functions
PcmsSetIdleChecker()	Set callback functions
PcmsSetAttrs()	Set attributes on a Dimensions object
PcmsCheckMessages()	Check results from Dimensions commands
PcmsGetConnectDesc()	Get current connection details

Using the *ptrEventInfo* in Events

The special void * pointer *ptrEventInfo* is filled in when certain events are fired to provide you with additional information pertinent to those events. The information contained in the pointer will change depending on the event which is being fired.

The following table lists the structures that this pointer needs to be typecast to depending on the event being fired.

DTK Event	DTK Structure
PCMS_EVENT_MAIL	PcmsUserRoleStruct
PCMS_EVENT_DLGC	PcmsUserRoleStruct
PCMS_EVENT_DLGI	PcmsUserRoleStruct
PCMS_EVENT_RELATE	PcmsRelStruct
PCMS_EVENT_UNRELATE	PcmsRelStruct

If you need to access the information in these structures, then your event needs to do the following:

- For RELATE and UNRELATE event types typecast the *ptrEventInfo* via:

```
PcmsRelStruct *ptrRel = (PcmsRelStruct*)*ptrEventInfo;
```

- For other events typecast the pointer via:

```
PcmsUserRoleStruct *ptrRel =
    (PcmsUserRoleStruct*)*ptrEventInfo;
```

Once you have typecast the pointer you can use the *ptrEventInfo* pointer to access the information. For example, in a MAIL event you might do the following:

```
{
   PcmsUserRoleStruct   *ptrUser =
        (PcmsUserRoleStruct*)*ptrEventInfo;
   int                  noUses  = *noEventInfo;
   int                  i = 0;

   for (i = 0; i < noUsers; i++)
        (void)fprintf(fd,"\nFound user : %s",ptrUser[i].user);
}
```

In Delegate Events (DLGI and DLGC) the *applyDeny* and *treeWalk* members of *PcmsUserRoleStruct* have special meanings that

relate to the option specified on the command line. These meanings are listed below.

Operation	applyDeny	treeWalk
/ADD	Y	Y
/DELETE	Y	Y
/REPLACE	Y	N

Event Examples

The release media contains a number of example events and makefiles to help you get started. These are contained in the `pcms_api/examples` subdirectory in the Dimensions installation directory.

Events - A Final Word and a Warning

Events are a powerful and versatile way of expanding on Dimensions rich functionality. They allow you to implement your own specific process controls and integrations with external applications. Used well, events can enhance both your working practices and use of Dimensions. However, if your events have not been written carefully, you do run the risk of affecting Dimensions functionality, especially if your event causes memory corruption. You are strongly advised to test any complex events thoroughly before deploying them, and if possible use a memory tracking tool to verify memory use.

A Known DTK Event Issues

In this Appendix

This appendix discusses a number of known issues that you should keep in mind when using DTK events.

Missing Events

Some operations are currently missing events being fired which you might expect to be fired. The Dimensions commands which are affected are listed below.

AC – Action Change Document	No VALIDATE event is fired.
CBL – Create Baseline	No POST_CREATE event is fired.
CMB – Create Merged Baseline	No POST_CREATE event is fired.
MWS – Merge workset	No POST_CREATE event is fired.

Index

A

C

D

E

F

G

H

I

M

O

P

R

S

T

U

V

W